



THE HONG KONG
POLYTECHNIC UNIVERSITY

香港理工大學

Pao Yue-kong Library

包玉剛圖書館

Copyright Undertaking

This thesis is protected by copyright, with all rights reserved.

By reading and using the thesis, the reader understands and agrees to the following terms:

1. The reader will abide by the rules and legal ordinances governing copyright regarding the use of the thesis.
2. The reader will use the thesis for the purpose of research or private study only and not for distribution or further reproduction or any other purpose.
3. The reader agrees to indemnify and hold the University harmless from and against any loss, damage, cost, liability or expenses arising from copyright infringement or unauthorized usage.

IMPORTANT

If you have reasons to believe that any materials in this thesis are deemed not suitable to be distributed in this form, or a copyright owner having difficulty with the material being included in our database, please contact lbsys@polyu.edu.hk providing details. The Library will look into your claim and consider taking remedial action upon receipt of the written requests.

The Hong Kong Polytechnic University

The Department of Industrial and Systems Engineering

**A Concurrency Integrity Model for
Distributed Product Data Management**

CHAN Edmond Cheuk Kit

A thesis submitted in partial fulfillment of the requirements for the
Degree of Doctor of Philosophy

January 2009

CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

CHAN Edmond Cheuk Kit

Abstract

In today's manufacturing environment, enterprises having work groups geographically dispersed are not uncommon. In addition, different tasks of the product lifecycle are distributed at different geographic locations. A product data management (PDM) system is therefore required for controlling the distribution and maintaining the integrity of the product data throughout its entire life cycle. Multiple accesses to the system residing on multiple sites will cause concurrency problems. It is crucial that concurrency control must be provided to protect the data against a variety of possible threats, in particular, data inconsistencies must be avoided and relationships between data must be maintained. In particular, the current PDM technology is not completely suitable for PDM in distributed manufacturing environment. Thus, this research aims to develop a foundation in concurrent engineering support for distributed product data management (DPDM).

This research begins with reviewing the suitability of existing database management systems for product data within the scope of data architecture. These systems are based on the technology for managing business data. In other words, they are not efficient when employed to handle product data. Therefore, an ontological approach for representing product data is proposed to describe the relationships between all the objects within a DPDM system.

To secure the consistency of a DPDM system, the functions of the system must be error-free. A DPDM system specification is necessary. Firstly, a graphical representation model is developed to express the PDM functions. UML (Unified Modeling Language) sequence diagrams are used to model the actions of each of the functions and their interactions between users and the system. Temporal logic is then used to construct the formula of these functions. The model is further extended to represent DPDM functions.

Given that the traditional concurrency control methods were not purposely developed to meet the need of DPDM, the requirements for data storage and

manipulation for DPDM systems are not well supported. Therefore, a concurrent control method that caters for version management and product architecture in DPDM is proposed. This research demonstrates how granularity and versioning are incorporated into a lock-based concurrency control model. The concurrent accessibility of an example product data is explained to illustrate the adjustability according to the actions taken by the users and the architecture of the corresponding data object.

Locking is one of the well-known concurrency control techniques and more likely to be encountered in practice. Although lock-based concurrency control methods guarantee serializability of data access, the systems have the risk of deadlock as the transactions may wait for unavailable locks. An integer programming based mathematical model employing transaction scheduling is proposed to prevent the threat of deadlock by controlling transaction executions in DPDM system while efficiency is maintained.

To validate the performance of the proposed methods, the strict two-phase locking (2PL) method, the two-phase granularity version (GV) locking method, the 2PL transaction scheduling method, and the hybrid GV transaction scheduling method are evaluated through simulation experiments. Read and Write accesses to composite objects are used to illustrate the comparisons between the models. Their performances are evaluated by comparing the number of late transactions. It shows that the GV locking model and the transaction scheduling model are better than the strict PL method while the hybrid model can substantially improve the concurrency of DPDM system.

Publications Arising From the Thesis

Chan, E. C. K. & Yu, K. M. (2007). A concurrency control model for PDM systems. *Computers in Industry*, 58(8-9), 823-831.

Chan, E. C. K. & Yu, K. M. (2007). A framework of ontology-enabled product knowledge management. *International Journal of Product Development* 4(3-4), 241 – 254.

Chan, E. C. K. & Yu, K. M. Conflict Avoidance using Integer Programming Transaction Scheduling for Product Data Management. *Computers in Industry*. (Under Review)

Chan, E. C. K. & Yu, K. M. A model for project-based environmental compliance management for SMEs. *International Journal of Product Development*. (Under Review)

Acknowledgements

First and foremost I would like to thank my supervisor, Dr K. M. Yu for his contributions towards the algorithm, implementation and presentation of this thesis. His comments regarding the applications of PDM system has been most useful. Also his insight and guidance he provided throughout the entire research period has been invaluable.

Finally, I would like to thank my family for their care and encouragement. I would never have completed my study without their supports.

*The fear of the LORD is the beginning of wisdom,
and knowledge of the Holy One is understanding.*

Proverbs 9:10

Table of Contents

ABSTRACT	i
PUBLICATIONS ARISING FROM THE THESIS	iii
ACKNOWLEDGEMENT	iv
LIST OF FIGURES	x
LIST OF TABLES	xiii
NOTATIONS	xiv
CHAPTER 1 INTRODUCTION	1.1
1.1 Background	1.1
1.2 Key Issues and Problems	1.3
1.3 Research Objectives	1.5
1.4 Significance of the Research	1.5
1.5 Organisation of the Thesis	1.6
CHAPTER 2 LITERATURE REVIEW	2.1
2.1 Introduction to Product Data Management System	2.1
2.1.1 Workflow and Process Management	2.2
2.1.2 Product Structure Management	2.3
2.1.3 Classification	2.3
2.1.4 Program Management	2.4
2.2 Data Model for PDM System	2.4
2.2.1 Object Oriented Approach	2.5
2.2.2 Object Oriented Technology for PDM System	2.5
2.2.3 Ontology	2.6
2.3 System Modeling and Specification	2.7
2.3.1 Graphical Modeling Tools	2.7
2.3.2 Formal Specification	2.11
2.4 Concurrency Control	2.13
2.4.1 Two Phase Locking	2.14
2.4.2 Granularity Locking	2.15
2.4.3 Version Locking	2.16

2.4.4	Flow Graph Locking	2.16
2.4.5	Timestamp Ordering	2.17
2.4.6	Deadlock	2.18
2.5	Limitations of Existing Approaches	2.20
2.5.1	Justification of Tools and Techniques Adopted	2.22

CHAPTER 3 ONTOLOGICAL DATA MODELING IN PDM SYSTEM 3.1

3.1	Application of Ontology to PDM	3.1
3.1.1	Procedure of Creating Ontology	3.2
3.1.2	Evolvable Ontology with Options of Instance	3.6
3.2	Mechanism of Ontology-Enabled PDM System	3.6
3.2.1	Notations of the Data Model	3.7
3.2.2	Definition of Data Category	3.8
3.3	Ontology Management Functions	3.9
3.3.1	Item Insertion	3.10
3.3.2	Ontology Creation	3.11
3.3.3	Ontology Retrieval	3.13

CHAPTER 4 ONTOLOGY-BASED ENVIRONMENTAL COMPLIANCE MANAGEMENT SYSTEM 4.1

4.1	Background on Environmental Compliance	4.1
4.2	Environmental Compliance Management System	4.2
4.2.1	Information Management Tools	4.3
4.2.2	Architecture of PDM System	4.4
4.3	The Model of Ontology-Enabled ECM System	4.6
4.3.1	Structure of the System	4.6
4.3.2	The Compliance Analysis Module	4.8
4.4	Implementation of ECM System	4.10
4.4.1	Software for Implementation	4.11
4.4.2	The Compliance Analysis Module – ECM System	4.11
4.4.3	Illustrative Example	4.12

CHAPTER 5 REPRESENTATION OF PDM FUNCTIONS IN UML SEQUENCE DIAGRAM	5.1
5.1 PDM User Functions	5.1
5.1.1 Description of PDM System User Functions	5.2
5.1.2 Description of PDM System Intrinsic Functions	5.3
5.2 Use of Sequence Diagram	5.4
5.2.1 Registering data in PDM System	5.4
5.2.2 Check-Out in PDM System	5.5
5.2.3 Check-In in PDM System	5.6
5.2.4 Release in PDM System	5.7
5.2.5 Obsolescence in PDM System	5.8
5.2.6 Deletion in PDM System	5.9
5.3 DPDM User Functions	5.10
5.3.1 Description of DPDM System User Functions	5.11
5.3.2 Description of DPDM System Intrinsic Functions	5.11
5.3.3 Registering data in DPDM system	5.12
5.3.4 Check-out in DPDM system	5.13
5.3.5 Check-In in DPDM System	5.15
5.3.6 Release in DPDM System	5.16
5.3.7 Obsolescence in DPDM System	5.17
5.3.8 Deletion in DPDM System	5.18
CHAPTER 6 CONCURRENCY CONTROL SPECIFICATION	6.1
6.1 Concurrency Problems in DPDM System	6.1
6.1.1 The Lost Update Problem	6.3
6.1.2 The Dependency Problem	6.4
6.3 Basic 2PL Protocol in Sequence Diagram	6.5
6.4 Formal Specification of Concurrency Control	6.7
6.4.1 Temporal Logic	6.7
6.4.2 Integration of Sequence Diagram and Temporal Logic	6.8
6.4.3 Representation of Serialisability in Temporal Logic	6.9
6.4.4 Specification of Two Phase Locking in Temporal Logic	6.9

CHAPTER 7	CONCURRENCY CONTROL METHOD FOR	
	DPDM SYSTEM	7.1
7.1	Formal Description of the Method	7.1
7.1.1	Locks with Version	7.2
7.1.2	Notations and Types of Functions	7.6
7.2	Implementation	7.7
7.2.1	Check-out and Release Processes	7.8
7.2.2	View Process	7.9
7.2.3	Obsolete Process	7.11
7.2.4	Function of Redlining	7.11
7.3	Case Study	7.12
CHAPTER 8	DPDM DEADLOCK AVOIDANCE	8.1
8.1	Transaction Scheduling Problem in DPDM System	8.1
8.1.1	Problem of Deadlocks	8.2
8.1.2	Definition of Transaction	8.3
8.1.3	Deadlock Avoidance	8.3
8.1.4	Objective of the Model	8.5
8.2	Set Partitioning Problem	8.8
8.2.1	Formulation of Concurrent Transaction Scheduling Problem	8.10
8.2.2	Solution Approach	8.14
CHAPTER 9	SIMULATIONS AND PERFORMANCE	
	EVALUATION	9.1
9.1	General Information of the Simulation	9.1
9.1.1	Event Oriented Simulation	9.2
9.2	Simulations and Results of Various Models	9.3
9.2.1	Two Phase Locking (2PL) Model	9.4
9.2.2	Granularity Version Locking	9.7
9.2.3	Transaction Scheduling	9.12
9.2.4	The Combined Model	9.18

CHAPTER 10 DISCUSSION	10.1
10.1 DPDM System Representation	10.1
10.2 DPDM Specifications	10.1
10.3 Granularity Version Locking Method	10.2
10.4 Transaction Scheduling Method	10.3
CHAPTER 11 CONCLUSIONS	11.1
11.1 Contribution of the Research	11.1
11.1.1 Data Modeling for DPDM system	11.1
11.1.2 Graphical and Logical Representation of DPDM Specifications	11.2
11.1.3 Granularity Version Locking for PDM/DPDM system	11.2
11.1.4 Transaction Scheduling Method for DPDM Deadlock Avoidance	11.3
11.2 Future Research	11.4
11.2.1 Standard Object Oriented Database Language	11.4
11.2.2 Utilisation of the capability of an ontology based PDM system	11.4
11.2.3 Automatic Translation from UML Sequence Diagrams to Formal Semantics	11.5
11.2.4 Relaxation of Two Phase Locking	11.5
11.2.5 Dynamic Programming for Concurrent Transaction Scheduling	11.6
REFERENCES	Ref.1
APPENDIX A EDRAWING PLUG-IN	A.1
Key Features of eDrawing Plug-in	A.A.1
Creating an eDrawing Files	A.A.1
Sending an eDrawing file	A.A.2
Markup Tools	A.A.2
APPENDIX B CONSTRAINT BRANCHING	1
Bounding Procedure and Termination Criterion	B.3

List of Figures

Figure 2.1	User accesses a PDM system through the system interface.	2.2
Figure 2.2	Bank Teller state transition diagram	2.9
Figure 2.3	Deadlock occurs when transaction are requesting data from each other	2.19
Figure 3.1	Framework of ontology-enabled PDM system	3.2
Figure 3.2 (a)	Assembly drawing of an ink cartridge holder	3.3
Figure 3.2 (b)	Drawing of an ink cartridge holder in exploded view	3.3
Figure 3.3	Create classes	3.5
Figure 3.4	Creating an instance of ink cartridge.....	3.5
Figure 3.5	Relationship of data classes of PDM system.....	3.8
Figure 3.6	Algorithm for inserting new item into an ontology.....	3.10
Figure 3.7	Algorithm for creating a new ontology	3.11
Figure 3.8	Creating new ontology in accordance with the object's complexity	3.12
Figure 3.9	Algorithm for ontology retrieval	3.13
Figure 4.1	Overview of the three-tier architecture	4.5
Figure 4.2	Model of environmental compliance management system ..	4.8
Figure 4.3	Procedure of compliance analysis	4.10
Figure 4.4	Start up page of ECM System	4.13
Figure 4.5	BOM display of the product being analysed	4.14
Figure 4.6	Data schema of part database in ECM System.....	4.15
Figure 4.7	Analysis on the grading of parts.....	4.16
Figure 4.8	Report showing the compliant status of the product	4.17
Figure 4.9	ECM System created analysis report in DXF format.....	4.18
Figure 5.1	Flow of data under the operation of various PDM functions	5.2
Figure 5.2	Sequence diagram for registering data	5.5

Figure 5.3	Sequence diagram for checking out data object.....	5.6
Figure 5.4	Sequence diagram for checking-in data object.....	5.7
Figure 5.5	Sequence diagram for releasing data object	5.8
Figure 5.6	Sequence diagram for data object obsolescence.....	5.9
Figure 5.7	Sequence diagram for deleting data object.....	5.9
Figure 5.8	Sequence diagram for registering data in DPDM system ..	5.13
Figure 5.9	Sequence diagram for checking out data object in DPDM system	5.14
Figure 5.10	Sequence diagram for checking-in objects in DPDM system	5.15
Figure 5.11	Sequence diagram for releasing objects in DPDM system	5.17
Figure 5.12	Sequence diagram for obsoleting objects in DPDM system	5.18
Figure 5.13	Sequence diagram for deleting objects in DPDM system..	5.19
Figure 6.1	Basic database operations: Read and Write	6.2
Figure 6.2	The lost update problem	6.3
Figure 6.3	Dependency problem: Referencing an undone data object ..	6.4
Figure 6.4	Dependency problem: Undoing the work of other transaction	6.5
Figure 6.5	Basic two phase locking protocol.....	6.6
Figure 7.1	Lock granules hierarchy	7.3
Figure 7.2	Illustration of lock compatibility	7.5
Figure 7.3	Check-out/Release process on part data object pd^r with one direct assembly	7.8
Figure 7.4	Check-out/Release process on part data object pd^r with more than one direct assembly	7.9
Figure 7.5	Viewing a part with more than one direct assembly	7.10
Figure 7.6	Versioning of redlining	7.12
Figure 7.7	The object class	7.13
Figure 7.8	Product structure of the ink jet printer.....	7.14

Figure 7.9 (a)	The assembly of pivot link	7.14
Figure 7.9 (b)	The part of ink cartridge latch	7.14
Figure 7.10	The design of the ink cartridge latch is being modified	7.15
Figure 7.11	The design of ink cartridge lid is being modified	7.16
Figure 8.1	Execution precedence graph.....	8.4
Figure 8.2	Outcome of executing transactions without scheduling.....	8.8
Figure 8.3	Illustration of integer programming for transaction scheduling problem.....	8.14
Figure 8.4	Proposed model of transaction scheduling for DPDM systems	8.17
Figure 9.1	Composition of data objects in database	9.4
Figure 9.2	Histogram of the late transactions in the basic model.....	9.7
Figure 9.3	Histogram of the tardy transactions in the granularity version locking model	9.10
Figure 9.4	Histogram of the tardy transactions in the transaction scheduling model.....	9.16
Figure 9.5	Histogram of the tardy transactions in the combined model	9.20
Figure A.1	Showing dimension of a hole using Dimension function.A.A.3	
Figure A.2	Adding comment to a document	A.A.3

List of Tables

Table 4.1	Structure of web technology and PDM system.....	4.6
Table 6.1	The truth table of repeat function to perform repetitive addition	6.12
Table 7.1	Compatibility matrix for granularity locking.....	7.4
Table 9.1	Attributes of transactions in system simulation example.....	9.2
Table 9.2	Example of event oriented simulation on PDM system.....	9.3
Table 9.3	Transaction examples of event orientated simulation of PDM system.....	9.5
Table 9.4	Simulation of two-phase locking with Wound-Wait and FIFO policy.....	9.6
Table 9.5	Average late transactions in the basic model.....	9.6
Table 9.6	Transactions in granularity version locking simulation.....	9.9
Table 9.7	Simulation example of granularity version locking in FIFO.	9.9
Table 9.8	Mean of late transactions of granularity version locking.....	9.11
Table 9.9	Summary of measures for comparing the basic model and the granularity version locking model.....	9.12
Table 9.10	Three transactions in the simulation of transaction scheduling method.....	9.15
Table 9.11	Simulation example of transaction scheduling method.....	9.15
Table 9.12	Mean tardy transactions of transaction scheduling model...	9.17
Table 9.13	Summary of measures for comparing the basic model and the transaction scheduling model.....	9.17
Table 9.14	Simulation example of combined model.....	9.19
Table 9.15	Mean tardy transactions of the combined model.....	9.21
Table 9.16	Statistical summary of performance comparison.....	9.22

Notations

db	Database
$PD = \{pd_1, pd_2, \dots, pd_n\}$	Product data object set
$AD = \{ad_1, ad_2, \dots, ad_n\}$	Assembly data object set
R	Read process
W	Write process
$\#sd$	Number of data objects d in the system
$\#ad$	Number of assemblies a
$\#pd$	Number of parts p
\emptyset	Empty set
\oplus	Exclusive or
\in	Member of
\subset	Proper subset
\rightarrow	Execute
\Rightarrow	Imply
$O = \{o_1, o_2, \dots, o_n\}$	Ontology set
s	PDM server
u	PDM user
Opt	Operation
T	Transaction
t	Time
U	Undo process
$\circ p$	Next: In the next moment in time that p will be true.
$\bullet p$	Previous: In the previous moment in time that p was true.
$\square p$	Henceforth: For all future time p is true.
$\blacksquare p$	Has been: from the preceding moment in time that p is true (including now)
$\diamond p$	Eventually: At some future time p is true.
$\blacklozenge p$	Once: p holds at some preceding position.

$p \cup q$	Until: p is always true until the time when q becomes true
$p \mathcal{W} q$	Unless: $(p \cup q) \vee (\Box p)$
$F = \{f_1, f_2, \dots, f_n\}$	Function set
$\Gamma = \{\phi_1, \phi_2, \dots, \phi_n\}$	Criteria set
ws	Workspace
$S(p)$	Lock part p in Shared mode
$rS(p)$	Release the Shared lock of part p
$X(p)$	Lock part p in Exclusive mode
$rX(p)$	Release the Exclusive lock of part p
$V(p)$	Lock part p in Versioned mode
$rV(p)$	Release the Versioned lock of part p
$IS(a)$	Lock an assembly a in Intent Shared mode
$rIS(a)$	Release the Intent Shared lock of assembly a
$IV(a)$	Lock an assembly a in Intent Version mode
$rIV(a)$	Release the Intent Version lock of assembly a
$d.RL^i$	Redlining (version i) of data object d
$\tau = \{T_1, T_2, \dots, T_n\}$	Transaction set
$T = \{d_1, d_2, \dots, d_n\}$	Data object required by transaction T
c	Cost of executing a transaction
$\cup P_j$	Union of partition j
A	Constraint matrix of linear programming
a_{dmT}	Coefficient of mode of lock m on data object d by transaction T
y_d	Auxiliary binary variable of data object d
M	Arbitrary large number
y	Binary variable

Chapter 1

Introduction

1.1 Background

Product data management (PDM) systems have emerged over the last two decade due to the increasing growth of “islands of automation” within an organization [Harris 1996]. These systems were originally in-house solutions of many large corporations who found their progress being seriously restrained by paper-based systems. Early PDM systems were designed to improve the management of initial release of the data to manufacturing process.

Nowadays there are many comparable products launched to the market contemporarily, resulting in very keen competition between manufacturing companies. In order to improve their competitiveness, manufacturers may need to produce complex products with more functions in addition to innovative design and better quality in a short time frame. However, complex products require multidisciplinary design teams to master the design and to comply with safety and environmental regulations. In addition, for many enterprises, the different tasks of the product lifecycle are distributed at different geographic locations. Traditional centralized PDM systems are not designed to provide a communication infrastructure for the whole project network. In response to these new challenges, PDM system enhances collaborative work by online access and electronic interchange of product data [Manji 1995]. The new generation PDM systems enable enterprises to conduct its business activities in a more efficient way via ingenious management of product information. PDM systems are no longer limited to managing only information created in the designing phase but in the entire product lifecycle.

With the advent of the internet- and web-based technologies, PDM systems can now be executed more effectively and efficiently. The efficiency and quality of design and manufacturing processes can be greatly improved by product

information sharing and visualization in the system. The development of web-based PDM system is essential for supporting collaborative design and manufacturing at geographically dispersed sites [Rezayat 2000, Yeh & You 2002, Zhang, et al. 2004]. The web-based PDM system facilitates the process of data exchanging and sharing in order to increase the throughput of product data transaction.

Heavy daily information flow among the design offices and production plants is anticipated when there are more accesses to the data of the PDM systems. Proper management and seamless integration are crucial to the success of the business. The lack of communication among different product development stages often causes data consistency problems in product lifecycle. These problems become more prominent when companies lowering their operation cost by locating their production plants and design teams in different countries. Also, each of the companies' departments implements their own information management systems such as Enterprise Resource Planning (ERP), Customer Relation Management (CRM), and other manufacturing information systems. In such case, a centralized PDM system lacks the ability to provide a collaborative working environment to all the involved parties. Therefore, the concept of concurrent engineering, integrated product and process development, and others are introduced [Chen 1997]. They are accompanied with the PDM system to manage all product related data and provide data retrieval for product design and production. Distributed Product Data Management (DPDM) system has been developed to provide a solution to the above problems.

In addition to providing functions of ordinary centralized PDM systems, DPDM systems are able to distribute product data to remote sites by breaking down the geographical boundaries between distributed sites over a computer network. Thus, the efficiency of the product development and production process can be improved when the information flow is orderly controlled in a collaborative working environment.

1.2 Key Issues and Problems

There are a number of commercial PDM systems and their extension, namely Product Lifecycle Management systems, such as Dassault's PDMWorks and ENOVIA MatrixOne, Siemens's Teamcenter, Product Dossier, PTC's Windchill, and several others vendors like Agile, Consensus, Smart Solutions, and Right Angle [Miller, et al. 1999]. These systems provide PDM basic functions. Nonetheless they have some limitations and problems for managing product data in the distributed and collaborative manufacturing environment. Some PDM systems are a further development of their past versions, which were designed in a time when modern computing technologies were not available. The newer versions claim they have collaborative operatability simply by integrating web-based technologies to provide data communication within an enterprise. The existing systems are designed for local application and to manage data sharing on one centralized database. In a distributed environment, each of the dispersed sites possesses a local PDM system to manage its own local data; it is not possible for these systems to share the product data in a consistent manner and to communicate with each other correctly without proper concurrency control. It is obvious that the existence of a computer network or a collection of data is not sufficient to form a distributed product data management (DPDM) system.

The major ERP providers, SAP and Oracle also provide tools specifically designed for DPDM as an extension module of their software applications. These state-of-the-art software applications often only address the needs of business data management. They are specific tools or technology applications that optimize only part of the production process. Moreover, the differences in the data architecture between business data and product data limit the efficacy of these systems in the distributed manufacturing environment. ERP system is designed fundamentally for managing numerical data, such as the inventory data, customer information, production line schedules etc. and this is not suitable for product data, which is consisted of both the physical data (e.g. specifications, images, and CAD drawings) and the meta-data. Therefore, these

conventional information management systems are not considered as a suitable choice for managing contemporary product data. Although customization of ERP system for specific needs is not uncommon, customizing an ERP package can be complicated. Most businesses will implement the best practices embedded in the acquired ERP system. Because of the special nature of the customization and the 'one off' aspect of the work, the cost of customization is expected to be high. Another factors that makes adoption of customized ERP as a DPDM system unfavourable is that the work delivered as customization is not covered by the ERP vendors maintenance agreement, there is no warranty the customizations would be inline with the next upgrade of the core product. Also, the customization may not be properly documented; new users may have difficulties in learning the customized module. Without the specification of the customization, the effort of the development will be wasted if someone who is responsible for system maintenance cannot remap the work to the updated system. A representation of DPDM system functions is needed to unify the underlying structures and the relationship of the product data and the system.

In a data management system, data model is important for the system design. Many PDM systems have adopted relational model of data as the underlying formalism due to their maturity and powerful features, consequently Relational Data Base Management System (RDBMS) has been chosen to be the cornerstone of PDM system architecture and design. This approach has overlooked the differences between the nature of product data and commercial data. In product development, different types of product data are generated and the data objects are usually very complicated. RDBMS is not efficient in managing large variety of data types and complex data objects. There is clearly a need for defining a data model of the product data in order to develop a robust DPDM system. Researches devoted to concurrency control (CC) in relational database indeed are not designed for handling distribution of product data, the application of these CC methods in PDM systems may yield unsatisfactory results and put data integrity in jeopardy.

1.3 Research Objectives

Given the shortcomings in implementing DPDM systems with the existing data management technology mentioned in the previous sections, this research focuses on bridging the gap by investigating the theoretical aspects of DPDM system. The specific objectives of this research are:

- i. To develop a data model specifically for product data management system, such that the representation of information about a product can be precisely defined.
- ii. To establish a specification of DPDM functions that enables product data to be correctly organized and maintained.
- iii. To develop a concurrency control model for providing deadlock free concurrent information and control flows to DPDM system.

1.4 Significance of the Research

This research aims to establish rigorous theoretical foundation in concurrent engineering support for distributed product data management. The research introduces ontology for modeling and specifying the data structure. This can concisely depict the relationships between products and basic components of which they are composed with related product data. This provides a data model for generic workflow which can respond efficiently to a heavy data exchange and sharing in a collaborative environment. In order to develop an error-free enterprise-independent PDM system, a generic representation PDM specification will be formulated in temporal logic. With the generic specification, the PDM system performance will no longer be tied up to any implementation tool nor the enterprise needs to adapt its workflow to suit only existing technology.

1.5 Organisation of the Thesis

The thesis consists of eleven chapters. The outline of the thesis is as follows:

Chapter 1: The problems that occur in distributed product data management are described and the objectives of the research are stated.

Chapter 2: Background and recent development of database management system are reviewed. These include introduction to PDM system and its functionalities and other data management systems, methods of system representation, and current techniques on maintaining data consistency.

Chapter 3: This chapter introduces a semantic data model by the use of ontology. The model attempts to facilitate query processing and integrity checking of PDM system model.

Chapter 4: The development of a framework in environmental compliance management (ECM) system by employing a PDM system and web technology is proposed. An ECM system is implemented to show how it helps a company to analyse the compliance of a product in the two directives, RoHS (Restriction of the use of certain Hazardous Substances in electrical and electronic equipment) and WEEE (Waste Electrical and Electronic Equipment).

Chapter 5: The representations of PDM system and distributed PDM (DPDM) system using UML sequence diagram and first order logic are presented. The advantages of integrating the graphical representation tool with formal notation are also discussed.

Chapter 6: This chapter specifies concurrency control for DPDM system using UML sequence diagrams and propositional temporal logic.

The approach is illustrated by specifying the two-phase locking method.

Chapter 7: A granularity versioning (GV) concurrent control model for PDM system that can also cater for version management and product architecture is presented in this chapter. A lock-based concurrency control model which utilises granularity and versioning to improve concurrency of distributed system is discussed.

Chapter 8: A transaction scheduling (TS) algorithm is proposed to eliminate the threat of deadlock in concurrency control. It discusses the use of integer programming based scheduling technique to control transaction executions in a PDM system.

Chapter 9: Simulation experiments are conducted to evaluate the performance of the proposed models. The capability of integrating various concurrency control methods with the TS algorithms is demonstrated.

Chapter 10: The limitations and the future research of the proposed approaches for system specifications and concurrency control for DPDM system are discussed.

Chapter 11: This chapter presents the conclusion and the contribution of this study.

Chapter 2

Literature Review

2.1 Introduction to Product Data Management System

The primary functionality of a PDM system is to provide a secure repository for storing product definition information and other functions in a PDM system will be defined in the later sections [Philpotts 1996]. A functional view of a PDM system is shown in Figure 2.1. A PDM system consists of a product database and a meta-database; users can only access the product data through the system interface and retrieve their information by querying the meta-database. Data controlled by PDM cannot be accessed without going through the proper PDM system's procedures and users can only store and retrieve data using check-in and check-out functions respectively.

The meta-database of a PDM system stores information of a product, so that its changes and list of authorised personnel can be tracked. Some information like the physical location of a data is hidden from users and applications; this procedure guarantees that there is no direct access to the product database. Hence, the data integrity can be maintained by monitoring and controlling all transactions between users and the system.

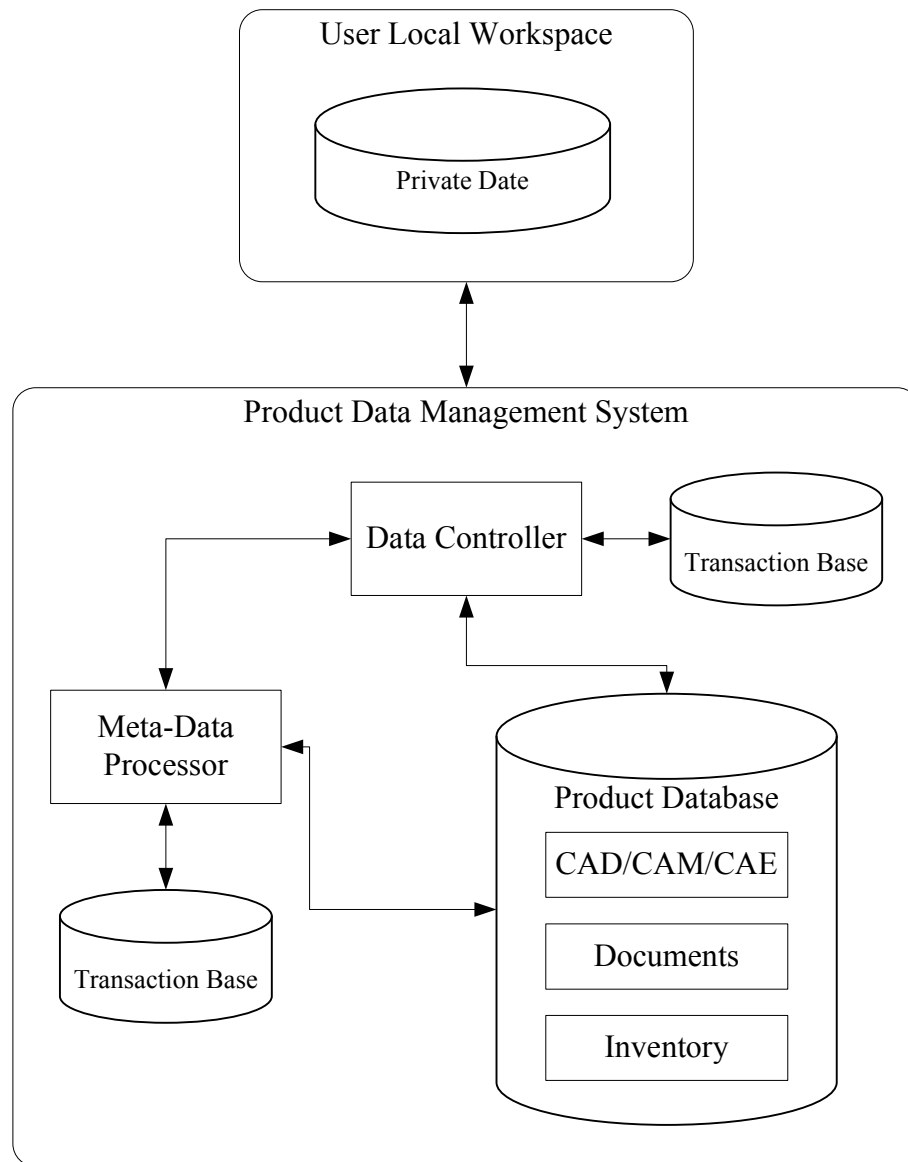


Figure 2.1 User accesses a PDM system through the system interface

2.1.1 Workflow and Process Management

A piece of product information passes through a sequence of processes before it can be used in the other stages of a product lifecycle. Product lifecycle management (PLM) is one of the cornerstones of a corporation's information technology structure. PLM is a systematic process for managing the entire lifecycle of a product from its conception, through design and manufacture, to service and disposal. By implementing PDM system to support PLM, organisations can work with the production information according to their

predefined business processes regardless of place and time. The workflow of product information is commonly divided into four stages: initiation, verification, approval, and release [CIMdata 1996]. These repetitive processes can be programmed within PDM systems, the objective is to ensure that all product information go through the predefined sequence of processes. The efficiency of the workflow can also be improved by workflow and process management as a PDM system can proactively progress a document to its next stage of process when the work in the current stage is completed.

2.1.2 Product Structure Management

In a PDM system, product data can be organised according to the product structure; the structure is usually determined by the relations between each component of a product. This facilitates users to determine easily which information will be affected when changes are made [Miller, et al. 1999].

2.1.3 Classification

PDM classification functions group parts, processes and other design information by common attributes [ASME 1998, SolidWorks 2005]. Searches for alternatives can be performed based on the values of attributes defined in the system for a particular item. Standard and similar parts can be found more easily, the product development time frame can be shortened when engineers and designers can re-use them instead of reinventing from scratch. Moreover, PDM systems facilitate capturing solutions from product development processes for future reference. For instance, a new designing approach for a new project may be potentially useful in other project. The approach can be documented and stored into the system and this allows people to look for solutions when they deal with similar problems.

2.1.4 Project Management

PDM systems provide project management functions to monitor project progress [Kim, et al. 1998, IBM Corporation 2007]. Projects are shown in work breakdown structure, so that the progress of tasks can be tracked easily. Completion of the product data required for each task is recorded against the plan, which enables users to see how a project proceeds in terms of the data's status.

2.2 Data Model for PDM System

A number of research issues must be addressed in order to develop an efficient concurrency control for PDM system. In designing and analyzing a PDM system, product modeling is an inevitable step. Product data model explains the relationships between components of a design and the various activities of a PDM system. In a data management system, data model is important for the system design. A semantic product data model was presented by Shaw, Susan Bloor, & Pennington [Shaw, et al. 1989] to support product design and manufacturing. Additional data modeling characteristic such as parameterization and data sharing have been introduced to support engineering design. Based on the semantic model in Chonoles and Quatrani [Chonoles & Quatrani 1996], Stadlbauer proposed a product data model for design support using functional skeletons [Stadlbauer 1992]. This feature represents the main functional flows in a product and allows the efficient storage of designs as well as the generation of verified products. A data model called Engineering Data Model (EDM) has been introduced by Pahng, Senin, and Wallace [Pahng, et al. 1998]. Their model is for representing design and engineering information, which defines a small set of structures capable of depicting a wide range of semantics necessary for engineering design.

In the early 80's, many relational database systems were introduced. Traditional RDBMS have been developed to meet the needs of business

applications, such as accounts, payroll, inventory control etc. These systems are based on the classical record-oriented data model that views data as a collection of relations, each having a collection of records stored in a table. The traditional database technology has several shortcomings in managing the data of new applications in computer-aided design and computer-aided manufacture. The environment of these next generation applications often requires long duration and cooperative transactions. There is a need for the ability to deal with complex data for computer-aided applications since the requirements of database system for product data management in term of both the data structures and the data model are very different to relational database systems.

2.2.1 Object-Oriented Approach

The technique of object-oriented for database system has emerged in two decades. Object-oriented (OO) methods organize both the information, and the process that manipulates the information to maintain a direct correspondence between real-world and database objects without losing their integrity and identity. In fact, OO database management systems (OODBMS's) provide more advantages over RDBMSs in many perspectives [Mansour, et al. 1995]. Researches in OODBMS have accelerated the move in designing information systems from conventional approach to OO. One of the characteristics of OODBMS is to provide the ability to describe the aggregation relationships between an object and objects of which it is composed for the purpose of storage and operation. The aggregation concept becomes the fundamental of versions [Chou & Kim 1986] and composite objects [Won, et al. 1989], which represent the version-of relationship and part-of relationship respectively. Various PDM systems today support these features.

2.2.2 Object-Oriented Technology for PDM System

PDM system architecture and design are also shifted from conventional approach to OO. For instance, Zhang proposed an open architecture [Zhang, et

al. 1995]. In particular, the components of the product data definition model, especially the principle and mechanism of the OODBMS are discussed. Architecture of Teamcenter Engineering of Siemens [Siemens 2008] also adapts a modular and object-oriented approach to provide a comprehensive framework to support the entire product life cycle. Rumbaugh's object modeling technique (OMT) has been adopted for the system design. Using the OO paradigm, an object-based data model is proposed for PDM system [Kim, et al. 1997, Liou 1994]. However, there is a limitation in the scope of the modeling that the model can only be used to manage drawings, parts, and product structure. To satisfy more requirements of real application, an extensible and general PDM system framework model is discussed [Kim, et al. 1998]. OO technology is applied to construct the generalized object model. However, the framework is based on RDBMS instead of OODBMS, the constructed object model cannot be mapped into RDB.

2.2.3 Ontology

Ontology has been limited to the study of philosophy in the past, it is now applied to a number of areas and its importance is recognised in many research fields. It is therefore necessary to clarify the intended meanings of the terms that will be used in designing PDM system. Firstly, the distinction between "Ontology" and an "ontology" is considered. The former is referred to as a subject of study in philosophy that is concerned with the nature of existence, and the latter is referred to as a logical theory accounting for the intended meaning of a formal vocabulary [Guarino 1998]. Bernaras, Laresgoti, and Corera [Bernaras, et al. 1996] stated that ontology is a mechanism that explicitly defines a domain with specifications of concepts, objects, relations and axioms. In other words, an ontology is a description of the properties of objects and the relations existing between different sorts of objects. Also, Patil, Dutta, and Sriram [Patil, et al. 2005] proposed an ontology-based framework to enable semantic interoperability between different application domains. A semantic equivalence matrix is introduced to resolve the ambiguities due to differences in meaning and syntaxes in different domains. Logic reasoning is

used to determine the semantic equivalences between application ontology and product semantic. Mapping of the results to the matrix is performed to determine the exact equivalent concepts.

An ontology can serve as a frame of reference for the discussion of the essential concepts of OO architecture. It has played a strategic role for developing object class [Eden & Hirshfeld 2001]; avoiding duplicate similar actions [Gruber 1993]; enumerating and standardizing important terms used [Chandrasekaran, et al. 1999]; and changing and updating legacy data in an effective way [Ding & Foo 2002]. Ontologies help people and computers to access the information they need, and effectively communicate with each other, since they describe the semantics of a domain in a way that humans can understand and computer can process.

2.3 System Modeling and Specification

In order to develop software that solves a particular problem, the desired properties needed to be achieved are usually written up in natural language. Such description is called a specification. From the specification, a model of the system can be built, which helps users to understand the reality and have computer simulations. Therefore, the specification should be defined as accurate as possible. However, it is widely agreed that a natural language cannot be considered as a good specification language. This is due to the fact that computers are not capable of understanding the meaning of natural language. The informality of such descriptions may cause ambiguities which could eventually result in serious flaws. So system analysts would try to define the problems to be solved by the software and generate a requirement analysis in a more formal approach.

2.3.1 Graphical Modeling Tools

Formal specification has to be unambiguous so that system developers can understand the requirements, develop a system that operates accordingly and

be able to verify that the specifications do not have any contradiction which would lead to inconsistency. These formal specifications are usually symbolic encodings of real-world constraints into some kind of logic. However, classical formal methods, namely mathematical proof, are not widely accepted in the industry since too many streams people can use to specify a system, although classical methods guarantee the correctness of a system without exhaustive tests [Drusinsky 2006]. A practical alternative of formal methods is graphical modeling. Graphical modeling languages are commonly used for specifying interactive systems and reactive systems. A few popular diagrammatic system analysis tools and their applications will be reviewed in this section.

The processes of systems can be modelled by Data Flow Diagram (DFD) that is introduced by Yourdon and Constantine [Yourdon 1979]. A DFD is a tool that shows how data enters and leaves a particular process. A network of processes is created by linking up the major activities of a software system with data flow paths. Formalism of DFD defined in graph theory [Sree, et al. 1990, Tao & Kung 1991] and formal semantics approach [Adler 1988, Gary, et al. 1999, Vazquez 1994] are proposed to make DFD models verifiable. Approaches for extending DFD for Modeling dynamic behaviours of real time systems have been presented in [Gomaa 1984, Kuo & Karimi 1988]. Examples of data flow-based systems are satellite image storage system [Abernethy & Kelly 1992] and an information system for a local health care agency [Farrell & Myers 1981]. One reason original DFD's are rarely used for system modeling nowadays is that they have not been standardized, since there are so many conventions. The other reason is that DFD's does not model time-dependent behaviour well, such as when processes are created or deleted. These make DFD's unsuitable for modeling event-driven systems of which events are supposed to be responded immediately when they occurred and processes have indefinable start and finish time.

A graphical modeling tool commonly used for defining event-driven systems is State Transition Diagrams (STD). STDs consist of a collection of nodes that represent states, connected by edges that represent state transition. State is the status that an object must be in before considering changing into another status.

Transition is an event that changes the state of an object to a specific state. Figure 2.1 is the state diagram of an automatic teller machine (ATM) [Langer 2008]. The boxes in the STD represent the possible statuses that exist in an ATM. The arrows are the events that trigger a change of state of an ATM. For example, when the ATM is in the “Enter Valid Card” state, the event of entering a valid bank card changes the state of asking for a valid card to prompting a password. At this state, if the bank card is invalid, the ATM will go to the state of “Enter Valid Card”, otherwise, it will ask the user to enter the password. The reason for using STD to model a system that never ends is because it can show the current state of a system and the ability to decide the state it goes to based on the input condition, unlike DFD that reflect only the data flow of a process. Therefore, STD is more suitable than DFD in modeling an on-going system that moving from one status to another without a definite end.

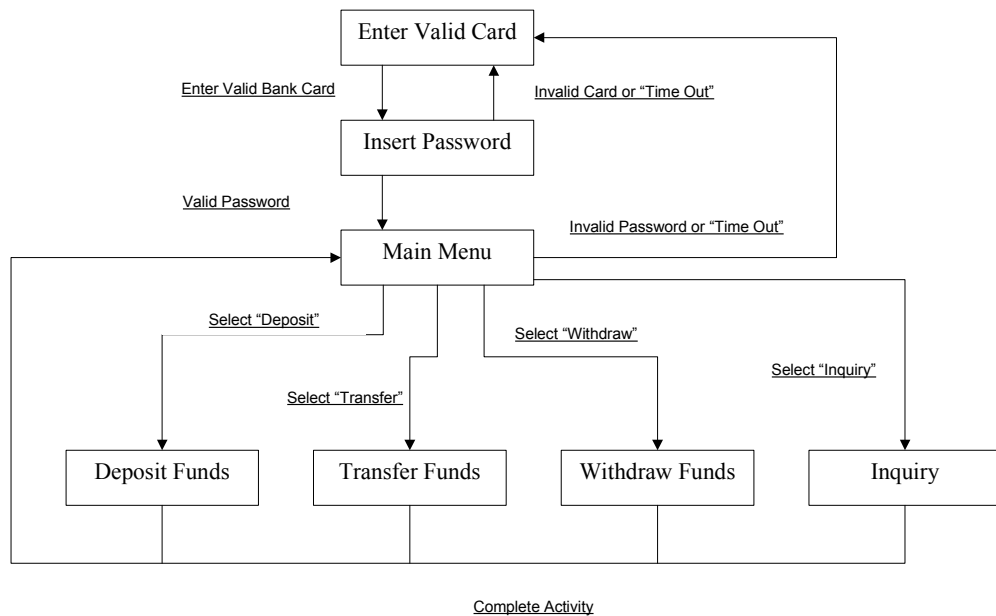


Figure 2.2 Bank Teller state transition diagram

Statechart is a variant of STD and have been used to support the design of interactive and automated systems. It extends STD with three techniques to enrich the modeling power expressiveness [Wieringa 2003]. In particular, parallelism allows statechart to represent concurrent processes in one diagram.

This technique is valuable for the design of DBMS's. Leong [Leong 2002] uses statecharts to model some of the DPDM system functions. Although statechart is a highly structured and economical description language for specifying system behaviours, like many other visual languages, the definition of the formal semantics of statechart has proved to be extremely challenging due to the richness of the language.

Unified Modeling Language (UML) [OMG 2007] is another popular graphical modeling tool to represent a system specification. UML was created with the goals of unifying the best features of different existing languages and of creating an industry standard. UML has been considered by many industrial and academic researchers as a promising system modeling language, because it is a semi-formal notation that is relatively easy to use and it is provided with code generation tools. However, the detailed system requirements are often overly simplified when being specified using standard template in natural language, thus system analysts may incorrectly interpret the UML function specifications and create many defects in the system development [Deepali, et al. 2005]. Furthermore, UML is not an executable specification language and there is no standard on how to validate such models, thus formal approach is needed for verifying UML-specified models.

In PDM system design, it is not uncommon to use various types of diagrams for different purposes. In the process-based PDM system approach [Chen & Tsao 1998], product development processes are described by flow charts, statecharts are used to describe changes to the objects and their relationship over time and the flow of data in processes are shown in DFD's to specify the data transfer between computers and servers in a PDM system. Later, a UML-based approach for implementing PDM system is proposed. The inter-relationship between data, retrieving method, and integration of the PDM into the product development process are of concern [Eynard, et al. 2004].

2.3.2 Formal Specification

In order to develop a system that works correctly, its functions need to be precisely described, such that people can examine their correctness, that is, the system can be tested whether it produces correct result generally and predicted with great accuracy under extreme conditions. The formal descriptions are referred to as specification. It is not uncommon that error-prone systems have a loosely-defined and incomplete specification. Precise specifications are essential for systems and the correctness is achieved by writing them in formal language. Logic is the tool chosen to establish the specification of a PDM system, as logic is a formal language for constructing arguments about situations in such a way that they can be reasoned formally. Prior to construction of a PDM system, it is worth to build a model in order to test that the design is correct. The benefit of this model is that it helps system designers to experiment the behaviour of the system operating under extreme conditions, thus avoid costly errors if unexpected events were happened.

A specification for describing the properties of a system can be represented with the formal language. Then the correctness of the system can be verified by checking the well formed logic description formulae that represent the system. As far as the formal notations are concerned, the classical logics, such as proposition and predicate logics, are not enough for DPDM specification. A system is classified as a reactive system if its role is to maintain ongoing interactions with its environment [Manna & Pnueli 1992]. Examples of reactive systems are communication networks, ATM machines, telephone systems, etc. Some reactive systems are not intended to terminate. Such systems have to be specified in terms of their continual behaviour. The main concern of reactive systems is that they do not operate in an orderly sequence of input, process and output. In many cases, a system may receive many new inputs from the environment, correctness of the systems becomes difficult to maintain as the number of interactions grow. As a result, time-varying change management is the standard. To cater for this feature, temporal logic deduction information management is a potential candidate.

Temporal logic was originally developed in order to represent tense in natural language. It is a well-developed branch of modal logic [Hughes & Cresswell 1968] and has been put forward by Pnueli [Pnueli 1981] and others as a useful tool for dealing with computer programs and digital hardware. It has been applied to the formal specification and verification of concurrent and distributed systems. For example, in the case study of knowledge reasoning [Dixon 2006], a knowledge game called Cluedo is specified using KL_n , which is a logic combining propositional linear temporal logic with a multi-agent technique. A system is developed for a player to find out the identity of other players in a game by inferring the past moves using a resolution base approach. Wood [Wood 1990] specifies the operation of elevators to demonstrate the appropriateness of temporal logic for system specification. An elevator model is built separately according to a subset of the logical formulas using the State Machine Language. Temporal logic is suggested to be a good representational tool for specifying concurrent systems. Also, a model of concurrent program executing n disjoint processes in a shared memory environment modelled in temporal logic is presented in [Pnueli 1981]. It demonstrates that specification of the nondeterministic behaviour of a program can be described using temporal logic.

Temporal logic is popular within computer science because it can formally specify the critical properties of systems, such as safety condition, liveness condition, and fairness condition.

Safety: These conditions are those that must not occur in the operations of a system. In the PDM system condition, for example, a transaction must lock a data object. Another example of a safety condition is that no more than one transaction give a write-lock to a data object.

Liveness: These conditions specify what the system must do. For example, whenever any transaction wants to lock a data object, it will eventually be permitted to do so.

Fairness: These conditions describe how nondeterministic specifications are to be resolved. For example, if a data object is free, and two transactions request the data object simultaneously by specifying the action to be taken, the fairness condition could express which transaction can have the priority to access the data object every time such a race condition occurs.

These conditions are typically expressed by giving a set of relationships enumerating the temporal constraints among events and actions. The temporal logic chosen to describe the concurrency model in this research is Propositional Temporal Logic (PTL) introduced by Pnueli [Manna & Pnueli 1992]. PTL is an extension of propositional logic with the additional temporal operators, and it does not permit explicit quantification on the variable time. It has been used for specification and synthesis of communicating processes [Manna & Wolper 1984]. Given that the database operations are event-based and the time of their presences are indefinite, PTL would provide an adequate expressiveness for specifying models without the complexity of the quantitative of time variable [Bellini, et al. 2000].

2.4 Concurrency Control

A PDM system consists of a number of components and the heavy flows of data within the system and between its users are expected. The data flows are further complicated within distributed DBMS's that support collaborative design and manufacturing at geographically dispersed sites. The web-based PDM system not only facilitate the process of data exchanging and sharing but also the number of transactions that access the database will be increased. A concurrent control mechanism is needed to coordinate concurrent accesses to a database to maintain data integrity.

A set of transactions can be executed serially or concurrently, a schedule is serial if all the database operations of one transaction are executed before any operation of the others. That is, the transactions are not interleaved. On the

other hand, if database operations from different transactions can be executed in parallel or interleave, a schedule is concurrent. Concurrency problems arise when there are two or more concurrent operations executed on a data object and at least one of which is a write operation. Such improper concurrent execution of a set of transactions violates the database consistency. As database consistency can be preserved by executing transactions serially. Therefore, if a concurrent execution of transactions is equivalent to any serial execution of those transactions, then the concurrent execution is serializable and it preserves the database consistency as well [Bernstein, et al. 1979]. Because of the serializability criterion, the database system needs to know only the sets of data objects whose access is required by transactions and their operations, it makes the serializability to be the well adopted correctness criterion for concurrent schedules.

In order to preserve the database consistency, some kinds of control mechanisms are clearly needed to ensure that concurrent transactions do not interfere with each other. This control, called concurrency control, manages a schedule of the database transactions, which is an arrangement of the execution of a transaction set. Since a schedule is correct if the execution of the set of transactions is serializable, the goal of database concurrency control is to ensure that all executions are serializable. Concurrency control is well studied in traditional DBMS's. However, there are relatively few studies that address this issue in PDM systems. Two most popular concurrency control mechanisms in traditional DBMS's will be reviewed in the following sections. In fact, many practical DBMS concurrency control algorithms are variation of the two basic techniques: locking [Eswaran, et al. 1976] and timestamp ordering [Li 1987].

2.4.1 Two Phase Locking

The two phase locking (2PL) technique is devised to control potential conflicts between read and write operations. It guarantees serializability by preventing a transaction from obtaining a lock on any data object after releasing it to another transaction [Bernstein, et al. 1979]. It requires each transaction to obtain a

read-lock or a write-lock on a data object before starting the reading or writing processes respectively. When a transaction requests a read-lock on a data object, this lock will only be granted if no other transactions have already held a write-lock on it. Similarly, a write-lock will only be granted if no other transactions have already held a read-lock or a write-lock on this data object. Once a transaction has acquired any lock, the transaction enters a growing phase. The moment a lock is released, the transaction enters a shrinking phase and is not allowed to acquire any more locks. The 2PL protocol is popular because of the ease of implementation due to its great simplicity. However, 2PL has the risk of deadlock as the transactions may wait for unavailable locks. Several 2PL-based techniques have been proposed to alleviate the deadlock problem for obtaining a higher degree of concurrency.

2.4.2 Granularity Locking

A product may consist of a single part, e.g. a screw, to as many as millions of parts, like a Boeing 747. The latter may structure into various systems, subsystems, assemblies and parts. However, PDM literatures seldom consider product architecture issue in the study of concurrency control. The product hierarchies should be used collaboratively with the concurrency control algorithms. A database can be organized as a hierarchy of lockable units [Carey 1983]. Granularity refers to the size of data unit that can be locked. The finer the granularity, the greater the concurrency; the coarser, the fewer locks to be set and tested [Jun 2000]. When a transaction sets a lock on a data object at a given level of hierarchy, it will implicitly lock all its descendents as well. The intention to lock at the higher levels of the hierarchy should be set before setting access lock at a lower level. A granularity locking applied to composite objects proposed by [Gary, et al. 1975]. However, this locking protocol does not recognise a composite object as a single lockable granule and may suffer from excessive overhead or restrictions on the data.

The lock overhead, data contention and resource contention are factors affecting the performance of different lock granularity [Ng & Hung 1995]. The

finer the lock granularity adopted, the more the lock overhead involved and the higher is the degree of both the data contention and the resource contention. When the transactions access the database sequentially, or the system is heavily loaded, coarse granularity is preferred for transactions accessing large number of data objects. When the transactions access the database randomly or the system is lightly loaded, fine granularity is a better choice for small or mixed transactions.

2.4.3 Version Locking

In product development, product designers will not only use the most recent version of the data object but also the previous one. Therefore, several versions of the same data object must be kept properly. A new version of data object is produced for each write on that data object. Version control helps to keep track of the evolution of the data objects being designed. It also allows rollback of changes made to file by storing the data corresponding to a context. Reisdorph [Reisdorph 1999] suggested using file history list to perform version control. A model of version is proposed by Talens, Chabane, and Colinas [Talens, et al. 1993] to facilitate the storage of version by avoiding information redundancy between successive versions. In addition, version control can be implemented with concurrency control, such as multiversion timestamp algorithm and multiversion lock-based with timestamp algorithm [Bernstein & Goodman 1983]. An adaptable constrained two versions two phase locking scheme is proposed for synchronizing the read and write lock request on the different versions of data [Goel, et al. 2000].

2.4.4 Flow Graph Locking

Flow graph locking (FGL) [Eich 1988] is a non-2PL locking method. This technique is based upon data flow graph and obtains serializability in execution of transactions as a product of data flow scheduling. A data flow graph is a directed graph where nodes represent operations to be performed and arcs

indicated scheduling constraints on the operations. Data flow along the arcs from operation to operation. The flow graph locking illustrated the data dependencies that exist between transactions, where the transactions are represented by the nodes with the progression of the locks directed by the arcs. This specialized data flow graph is called database flow graph. The FGL guarantees deadlock free and serializability. However, a transaction may have to lock data items that it does not access, which will increase locking overhead and lengthens the waiting time. Additional non-2PL protocols have been proposed for data organized as directed acyclic graphs (DAG) [Kedem & Silberschatz 1979, 1980, Yannakakis, et al. 1979]. These methods restrict the order in which data items can be locked based upon the graphical structure of the data. The DAG methods may require more data items to be locked than would be required with 2PL. Some methods may incur cascading rollbacks to ensure serializability. A cascading rollback occurs when the termination and rollback of one unfinished transaction causes the termination and rollback of other unfinished transactions.

2.4.5 Timestamp Ordering

Timestamp ordering (T/O) [Li 1987] technique assigns to each transaction a unique identifier which is its start time. When a transaction tries to issue a read-lock or write-lock on data object d , $read(d)$ or $write(d)$, the algorithm will compare the timestamp TS of $read_TS(d)$ and $write_TS(d)$ to determine which is the oldest timestamp among all timestamps of transactions that have read data object d successfully and the oldest timestamp among all the timestamps of transactions that have written d successfully. This ensures that the timestamp order of transaction execution is not violated. During a read action, if $read(d)$ of a transaction T with timestamp TS is younger than $read_TS(d)$, the $read(d)$ request is rejected and transaction T is aborted, else it is executed. If the order is violated, then transaction T is aborted and resubmitted to the system as a new transaction with a new timestamp. During a write action if $write(d)$ of a transaction T with timestamp TS is younger than

$\max\{read_TS(d), write_TS(d)\}$, the $write(d)$ request is rejected and transaction T is aborted, else it is executed. An aborted reaction is restarted with a new timestamp. This technique ensures deadlock free. However, there is a possibility of restarting and blocking if a transaction is continually aborted and restarted. Using timestamp and the knowledge of readsets and writesets, a deadlock free concurrency control scheme has been proposed by [Dasgupta & Kedem 1983]. The Delay/Reread protocol achieves consistency by requiring some write actions to be delayed and some reads to be reread [Mohan, et al. 1985]. Major drawbacks to this method are the overhead and requirement that some data are read twice.

2.4.6 Deadlock

Locking is one of the well-known concurrency control technique and more likely to be encountered in practice. The benefit of locking is the absence of cascading rollbacks. However, 2PL has the risk of deadlock as the transactions may wait for unavailable locks. Although locking guarantees serializable schedules, it is not necessarily deadlock free. Deadlock is a situation in which two or more transactions are waiting for data objects that are locked by others. Locking protocols can be modified to avoid the occurrence of deadlock. Deadlocks must not exist to ensure that every transaction will eventually be executed. The main approaches for resolving deadlock are either deadlock detection or deadlock avoidance.

Data dependency of the transactions can be determined by finding a cycle in a Wait-For Graph. Deadlock occurs when a cycle is formed among a set of transactions. An occurrence of a deadlock between two transactions, T_1 and T_2 , using a graph is illustrated in Figure 2.3. Trees are used to represent transactions, the starting node of a tree is labelled with the transaction name, other nodes are the data that locked by the transaction. Undirected arcs in a tree represent the relations between data and the transaction and directed arcs depict the request of data issued by the transaction, where the tail of the arc is at the

starting node of a transaction which makes the request and the tail is pointed to data which is being requested. A cycle exists when the starting node and the end node of a path are the same. It shows that the transactions that are included in the cycle are said to be deadlocked. Suppose T_2 does not need data object d_1 to process, T_1 will be the only transaction waiting a data which is locked by other transaction, then only arc a_1 is present in the graph and no cycle is formed, likewise for the presence of a_2 only. Deadlocks occur when concurrency control using locks is implemented to a PDM system. Common approach of selecting the transaction to be restarted comes in two versions, called Wound-Wait and Wait-Die [Bernstein & Goodman 1983]. Both versions determine the action upon T_1 if it is older than T_2 when T_1 requests a lock on a data that is already locked by T_2 . T_1 will be rolled back in Wound-Wait and will be waiting until T_2 completes its operations in Wait-Die.

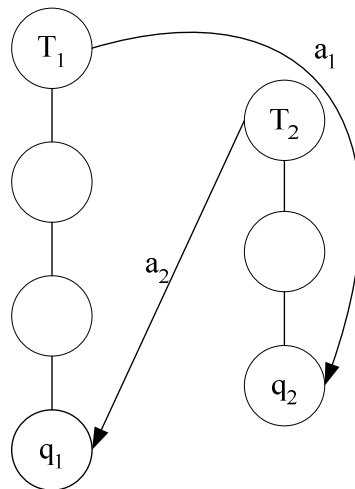


Figure 2.3 Deadlock occurs when transaction are requesting data from each other

2.5 Limitations of Existing Approaches

Concurrency control has been studied in many database applications. In particular, many works have been devoted to *real-time database management system* (RTDBMS). Applications of RTDBMS can be found in network management systems, military command and control management, and program trading in the stock market. Abbott and Garcia [Abbott & Garcia-Molina 1992] identified that conventional database systems do not emphasize the notion of deadlines for transactions. The concurrency control schemes designed for these databases lacked the ability to determine an execution schedule based on the time constraints of individual transaction. Various scheduling algorithms have been developed to schedule real-time tasks to meet their timing constraints [Charles 1982, Liu & James 1973, Zhao, et al. 1987], given the arrival time, deadline, estimated processing time, and priority of each task. Yu [Yu, et al. 1994] states that “traditional real-time scheduling usually does not address the data consistency issue, whereas consistency may have to be maintained by the concurrency control in database systems.” In addition to meeting the deadline requirements, there are other differences exist between conventional database and DPDM.

An adoption of any of these methods would not be an ideal solution for resolving concurrency problems of a DPDM system. Design and manufacturing workflow can be streamlined by implementing suitable DPDM system that manages all product-related data in an organized manner. One major function of DPDM system is to maintain data integrity and to provide accurate data when required [Leong, et al. 2003]. Above all, the differences between the natures of DPDM systems and conventional database systems limit the efficacy of these methods. Timestamping method is described as an optimistic scheme, since it assumes that conflicts are unlikely to happen in practice. One advantage of the scheme is that no updates are ever written to the database prior to successful completion of commit processing, so such restarts do not require any updates to be undone. The major drawback of the scheme is that restarting transactions waste all the effort which has already been spent on

the works. The effect of restart on databases somehow depends on the nature of the files. For example, a drawing design is being simultaneously altered by two CAD engineers, since this technique allows multiple accesses to data and performs checking at commit time to see whether a conflict did in fact occur. Hence, it is definitely that there is one engineer will have to redo his/her work if the work is submitted later than the other one. Many previous works use transactions rollback as a means for preserving consistency and deadlock freedom. These rollbacks require a considerable amount of overhead, and therefore degrade performance of the system. This performance cost had been considered acceptable since concurrent database systems in the past had a few transactions concurrently active. With the advent of network technologies, a number of machines accessing global databases and other resources is enormous. The amount of concurrency in a typical PDM system can be expected to rise dramatically. In such an environment, the use of rollbacks as a means for preserving consistency will become more burdensome.

On the other hands, locking schemes are considered pessimistic; as they assume that every piece of data accessed might be needed by other transactions and therefore better be locked. The effect of acquiring the lock is to prevent other transactions from changing the data objects in question. Despite 2PL outperforms the basic T/O and FGL in most cases [Carey & Livny 1989, Thomasian 1998], 2PL is not completely suitable for concurrency control in PDM systems. For example, if a DPDM system adopts 2PL technique as its concurrency control algorithm, each DPDM user must make sure that he/she can lock all the files that are required to be worked on that day. This is because other users will only release their files until the work completed under the 2PL technique.

One factor that influences DPDM concurrency is the data complexity. Among the functions of the DPDM system, concurrency control is essential to the checkout, release, obsolete, view, redlining, and references. In order to provide the control over the data access, most DPDM systems establish a set of access rules that determine what data can be accessed, in what mode and at what point in the product life cycle. This is complicated when a part or drawing belongs to

a certain assembly. In such instance, the modification to the assembly can cause the part to be locked out for write access until the process is completed. The more number of components of a data object has, the data object is more likely to be amended by more users. Locking all the data related to the data objects for making changes reduce the degree of concurrency or overall system throughput. A procedure that can isolate only the affected parts of the data will be helpful to conduct DPDM with efficiency. Therefore, a concurrency control should be specifically designed with the consideration to the unique characteristics of DPDM systems.

2.5.1 Justification of Tools and Techniques Adopted

In order to develop a concurrency control model for DPDM system, the meanings of objects and functions of a DPDM system and their relations must be precisely described. An ontology for DPDM system is developed in the research. Since by representing the semantics of data in a machine-processable form, ontology based reasoning service can provide consistency checking to the systems with respect to queries and assertions using the semantics defined in the ontology.

The proposed specifications of the PDM and DPMD functions in this research are developed using UML sequence diagrams. Sequence diagrams facilitate the specification process by allowing visual iteration through the operations, and also possess the expressiveness for both sequential and parallel operations. The diagrammatic specifications are then described in First Order Logic (FOL) and Propositional Temporal Logic (PTL). However, specifying DPDM functions in FOL alone is not sufficient to express the continual behaviour and time-varying changes of a dynamic system, and PTL is a complement to a formal specification for DPDM system.

Granularity Version locking proposed in this research is developed based upon the concurrency protocol proposed by Gary [Gary, et al. 1975]. Because 2-PL protocol never forces a transaction to be rolled back and never requires data to be reread. Version and Intent Version locks are introduced to lock the current

version of the data object at part and assembly level respectively, this allow the data object to be readable by other users while a new version is being created. The avoidance of deadlocks is achieved by scheduling the execution order of transactions subject to the compatibility of locks applied on data objects using integer programming.

Chapter 3

Ontological Data Modeling in PDM System

3.1 Application of Ontology to PDM

General concepts, such as actions, time and items can be formalized with reference to ontologies that are explicit specifications of conceptualisation. A method of design that based on generic ontology is presented in [Garcia, et al. 2004]. The significance of this effort is that it creates an ontology that specifies attributes and functional requirements of elements defined in a model. The case study of project development showed that the participants can work more effectively and have a high level of mutual consistency by using the project with ontology. Ontologies are needed to interpret the common understanding of structure of information among people and to enable reuse of domain knowledge. They provide a common vocabulary of an area and define - with different levels of formality - the meaning of terms and relations between them [Bansler & Havn 2003]. They also maintain the consistency of the system by guiding individuals' perception of products in accordance with the formal definition of ontologies.

An ontology that represents elements of the product development process forms the basis of the PDM system. The framework of an ontological PDM system that integrates an ontology-development tool, Protégé [Noy & McGuinness 2003], with a commercial PDM system, PDMWorks [SolidWorks 2005], is shown in Figure 3.1. The objective of creating an ontology-enabled PDM system is to create a repository for managing the definitions of objects for product development. By collecting data and information from various projects involved in the product development process, users can use the ontology-development tool to categorize this information using formal semantics for developing specification of concurrency control in PDM systems.

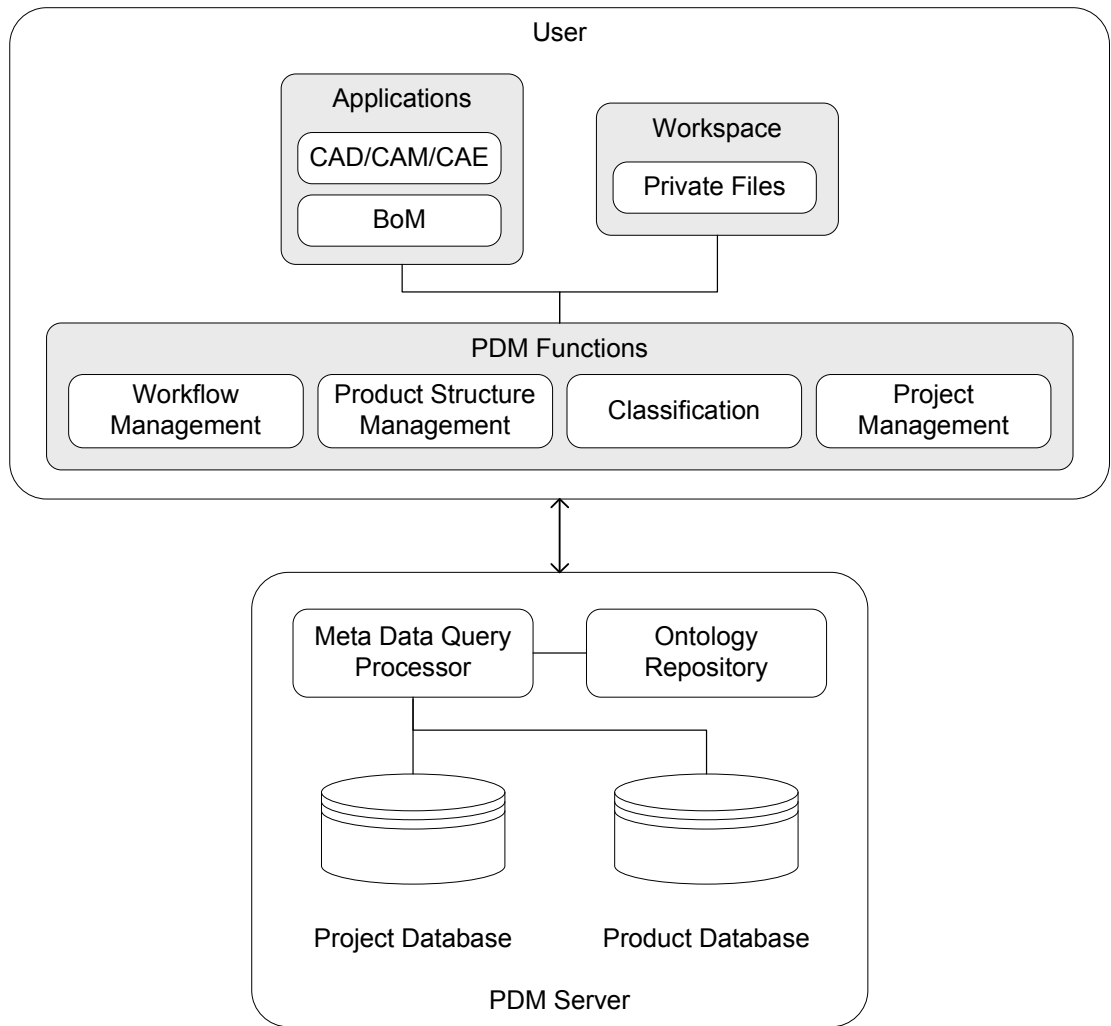


Figure 3.1 Framework of ontology-enabled PDM system

3.1.1 Procedure of Creating Ontology

A design of an ink cartridge holder of a printer from PDMWorks [Corporation 2004] shown in Figure 3.2 is used as an example to illustrate the procedure of developing ontology for the product development process using Protégé. The description of the six phases of the ontology development process is listed as follows:

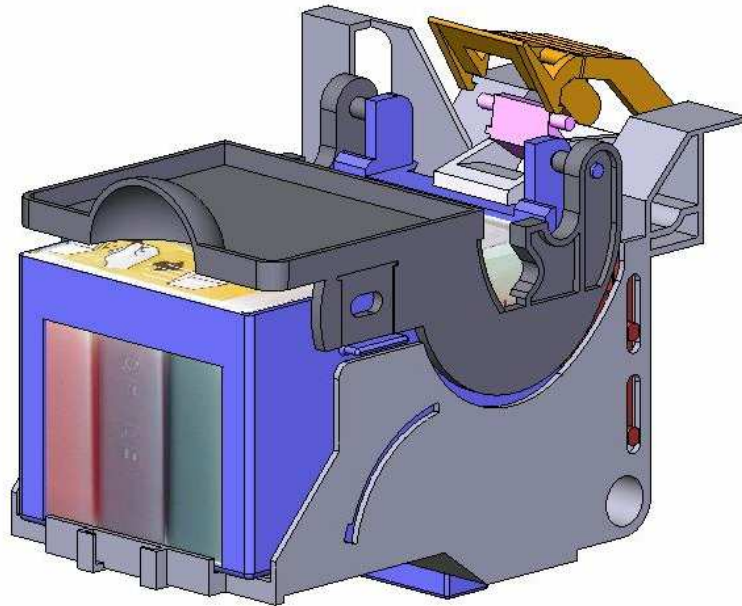


Figure 3.2 (a) Assembly drawing of an ink cartridge holder

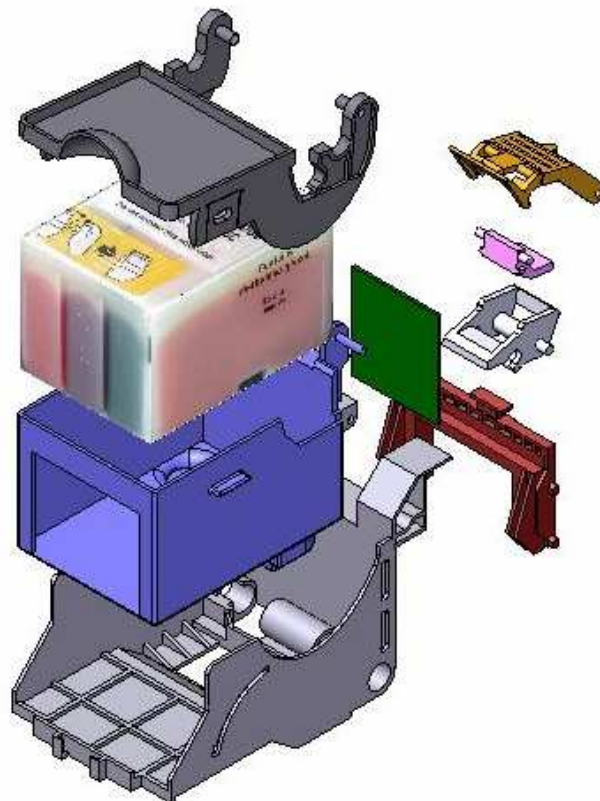


Figure 3.2 (b) Drawing of an ink cartridge holder in exploded view

1. *Determine Scope*

Determine the types of information that the ontology should provide temporal best practice and the domain that it should cover and the method of collecting the data from the PDM system.

2. *Enumerate Terms*

Demodularize the product into atomic parts and itemizes the important terms like “Cartridge Lid”, “Latch”, and “Pivot Link” and their properties are essential in standardizing the creation of classes (Figure 3.3).

3. *Define Classes*

Define classes for storing entities with similar characteristics or functions.

4. *Define Properties*

Define properties such as dimensions and colour of an object.

5. *Define Constraints*

Describe the set of possible values like minimum and maximum values for a slot is defined.

6. *Create Instances*

The class becomes a direct type of the instance and slot values are assigned to the instance frame (Figure 3.4).

Normally, the ontology development process in reality is more complicated than what has just been described. It often turns out to be a lengthy iterative process that involves repeatedly going through the phases in arbitrary order, except the phase of *determine scope*, before the ontology is formalized.

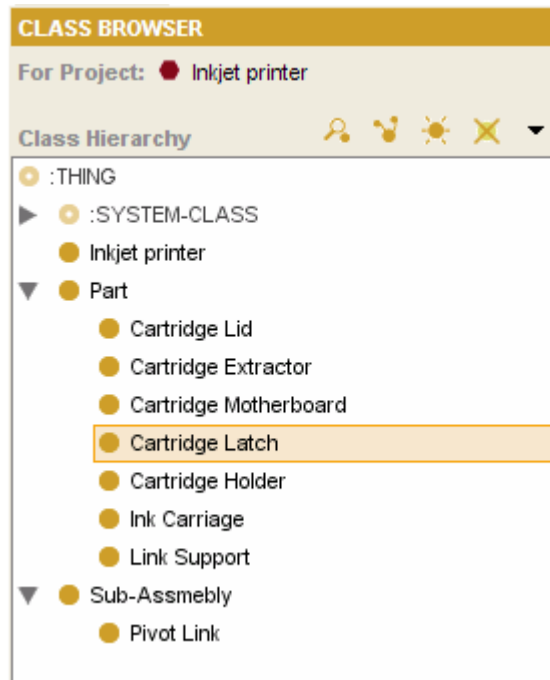


Figure 3.3 Create classes

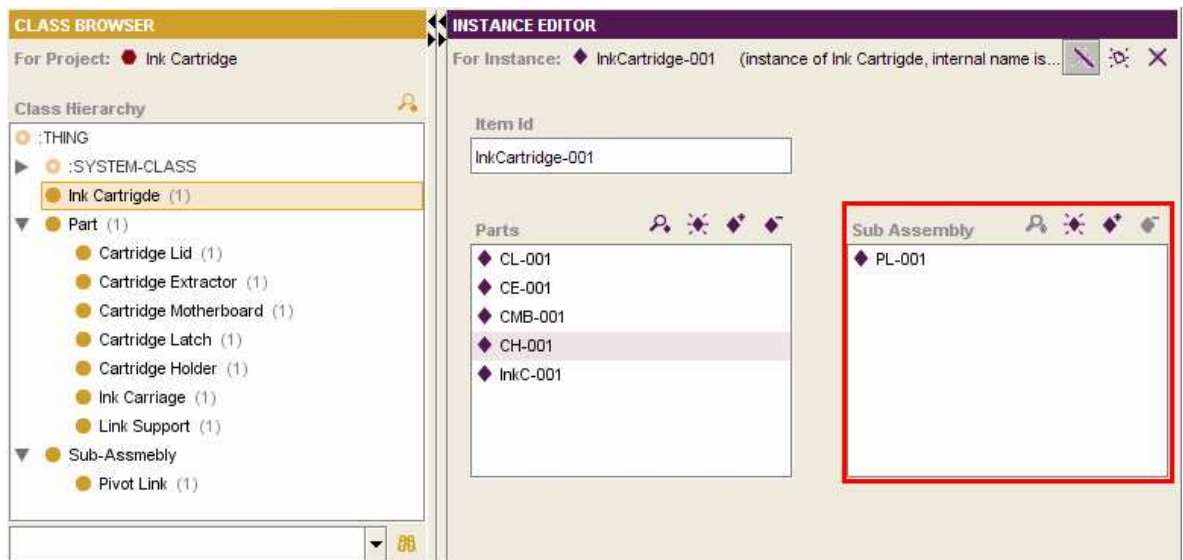


Figure 3.4 Creating an instance of ink cartridge

3.1.2 Evolvable Ontology with Options of Instance

Ontology-enabled PDM provides users with the flexibility in changing the specifications of a product. Once the ontology of a product has been formalized, the users need not go through the whole designing process to modify the product design; they can start at a specific phase that best describes the work nature. For example, if the colour of a cartridge latch needs to be changed, the users start at the fourth phase to change the latch with the right colour and directly go to the sixth phase to create a new instance. Certainly, users need to go through more phases when more modifications have to be made. For instance, the engineers would like to alter the dimensions of the ink carriage. The number and size of the components ink carriage can hold are also affected. Because the constraints of the object have to be redefined, the engineers have to start the redesign process from the fourth to the sixth phase of the development process.

3.2 Mechanism of Ontology-Enabled PDM System

PDM systems and ontology are tools for storing product data and reasoning behaviour across domains and projects. As mentioned in previous sections, the aim of this work is to design an implementation tool for building an ontology repository using information supplied by users. The framework lets users organise their works within a company in the form of a project, and this brings up many excellent features such as high flexibility, interdisciplinary work, and promoting innovation [Disterer 2002]. However, it is likely that groups of people would work on the same set of data objects concurrently, thus it is desirable to have a system that is capable to manage the access to the product ontology.

The organisation of physical objects into categories is a vital part of developing a PDM system. Categories serve to provide a sufficient description of relationship between system components and the data of physical objects. This

allows the system developer to define the specification of the system using formal semantics. First order logic (FOL) has been selected to discuss the content and organisation of ontology. FOL makes it easy to state facts about categories, either by relating object to categories or by quantifying over their members. Certain aspects of the real world are hard to capture in FOL, for example the temporal relationships between objects, their changes over time and sequences of operations of the PDM system. Although the ability to handle dynamic behaviours of the system and data objects is very important, it is better to lay down the most general definitions and postpone the discussion of time dependency until Chapter 5.

3.2.1 Notations of the Data Model

This section describes the notations used throughout the study. More notations will be introduced in later sections. The notations introduced here are for describing the sets used to make the formula more concise and readable.

The sets in the systems are:

The product database: db

Set of actions on the dataset: $Opt = \{R, W\}$, where R and W are the *read* and *write* actions on the dataset respectively.

Set of data class: $db = \{PD, AD\}$, where $PD = \{pd_1, pd_2, \dots, pd_m\}$ is the part data objects set and $AD = \{ad_1, ad_2, \dots, ad_n\}$ is the assembly data object set. PD is a subset of the database db and AD is a proper subset of PD . The Venn diagram below depicts the relations of the sets in the PDM system.

$$AD \subset PD \text{ and } PD \subseteq db$$

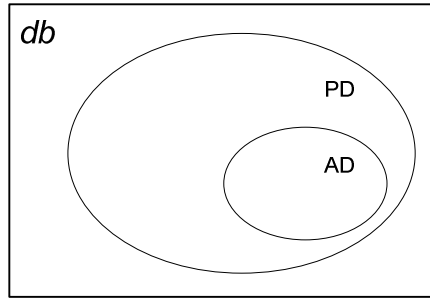


Figure 3.5 Relationship of data classes of PDM system

3.2.2 Definition of Data Category

The total number of data object in PDM system $\#sd$ is the sum of the components of which the database db consists.

$$\#sd = \#pd + \#ad \quad (3.1)$$

The number of components in each category cannot be a negative value.

$$\#pd, \#ad \geq 0 \quad (3.2)$$

For data object d to be a part, it must consist of only one data object

$$d \in PD \Rightarrow (\#d = 1) \quad (3.3)$$

For a data object d to be an assembly, it must be composed of at least two objects in either data classes and it cannot include itself as one of its components. The object is built of two sets of objects, where α is a set of assemblies and β is a set of parts.

$$d \in AD \Rightarrow d = \{\alpha, \beta \mid \#\alpha + \#\beta \geq 2\}$$

$$\text{where } \alpha = \{\emptyset\} \vee \{ad_1, \dots, ad_n \mid \forall i a_i \in AD\} \wedge \alpha \notin d$$

$$\beta = \{\emptyset\} \vee \{pd_1, \dots, pd_n \mid \forall i p_i \in PD\} \wedge \beta \notin d$$
(3.4)

Since all objects in the database must either be a part or an assembly, where \oplus is the symbol of exclusive-or,

$$\forall d \in db \quad d \in PD \oplus d \in AD$$
(3.5)

Thus, all data objects in the database must consist of at least one component.

$$\forall d \in db \quad \#d \geq 1$$
(3.6)

Based on (3.4), an assembly is composed of a number of parts and assemblies, the relation from A into P is:

$$AD = \left\{ ad_1, \dots, ad_n \mid \forall i ad_i = \{pd_1, \dots, pd_k\} \subset PD \wedge \left(\sum_1^k pd_i < \#pd \right) \right\}$$

$$\therefore AD \subset PD$$
(3.7)

3.3 Ontology Management Functions

In order to accomplish the integration of ontology engineering and DPDM system, three basic rules for managing the ontologies through the interface were developed. The rules and their corresponding algorithms are listed as follows:

The terms being used in the framework are:

$c(d)$: class to which data object d belongs

d_t : data object d at time t

$v(d)$: version of data object d

The atomic sentences that state facts in the framework:

$insert(d, o_i)$: object d is inserted into ontology o_i
 $retrieve(d, o_i)$: object d is retrieved from ontology o_i
 $return(d)$: object d is returned
 $p(d, a_i)$: d is a part of assembly data object a_i

3.3.1 Item Insertion

Let g be the item to be inserted to the ontology o_i in the ontology set $O = \{o_1, o_2, \dots, o_j, \dots, o_n\}$, which contains n ontologies. For g to be added, the class of o_i and g must be the same and no item in o_i is the same as g , the rule for item insertion is defined with a form like the following and the algorithm of item insertion is stated in Figure 3.6

$$p(d, o_i) \rightarrow insert(d, o_i) \quad (3.8)$$

```

begin
  /*Search for the ontology  $o$  in  $O$  and add  $g$  into  $o$  if exists*/
  while  $i < m$  and  $insert = FALSE$ 
    if  $o$  exists and  $x$  does not exist in  $o$  then
      insert  $d$  into  $o$ 
       $insert = TRUE$ 
    end-if
  end-while

  /*Create new ontology for storing object  $d$ */
  if  $insert = FALSE$  then
    ontology_creation( $o$ , subject, keywords)
    insert  $g$  into  $o$ 
  end-if
end

```

Figure 3.6 Algorithm for inserting new item into an ontology

3.3.2 Ontology Creation

Let n be the ontology that is intended to be added to the ontology set O ; $class$ be the class of a subject to which the ontology belongs and $keywords$ be a set of words or sentence that best describes the nature of the ontology. The $class$ k identifies the domain to which an ontology belongs, $keywords$ are used to search the depth level at which the new ontology is supposed to be in an ontology set. The rule of creating a new ontology in an ontology set is defined as follows and the algorithm of this rule is shown in Figure 3.7.

$$\neg \exists d_m \in o_i \quad c(o_i) = c(d_n) \wedge d_m = d_n \rightarrow insert(d_n, o_i) \quad (3.9)$$

```

begin
  /*use the keywords to determine the level at which  $o$  should be*/
  if  $o_n$  is inserted to the current level of the hierarchy then
    match = TRUE
    add  $o_n$  to the current level
    /*Search the hierarchy of the existing ontology according to the subject
    specified*/
    while  $i < m$  and match = FALSE
      if relation between  $o_i$  and  $O \neq$  relation between  $o_n$  and  $O$  and
        class( $i$ )  $\neq$  class( $n$ ) then
          /*Recursion – the algorithm searches down the hierarchy of the ontology
          until there is a match*/
          ontology_creation( $o_i, o_n, keywords$ )
          match = TRUE
        end-if
       $i = i + 1$ 
    end-while
  end-if
end

```

Figure 3.7 Algorithm for creating a new ontology

In the proposition of the ontology creation, o_y can be the ontology set or an ontology at any level that is a predecessor in the relationship with the new ontology o_x . For instance, the relationships between the objects are described in the following sentence: “*Cartridge lid is installed in an ink cartridge, which is an assembly that is installed in an inkjet printer*”, where the hierarchical relationship between the objects in the sentence is illustrated in Figure 3.8. Suppose item *cartridge lid* is to be added to the ontology set, the sentence shows that *cartridge lid* is part of an *ink cartridge*, which is part of an *inkjet printer*. Therefore, the ontology of *inkjet printer* is the predecessor of *ink cartridge*'s and *cartridge lid*'s ontologies. The ontologies at the upper levels of the hierarchy can be refined into a number of ontologies, they are the description of objects that are constituents of others. The process of ontology creation stops when the physical meaning of the further refinement of an object is not of any interest or importance to the users.

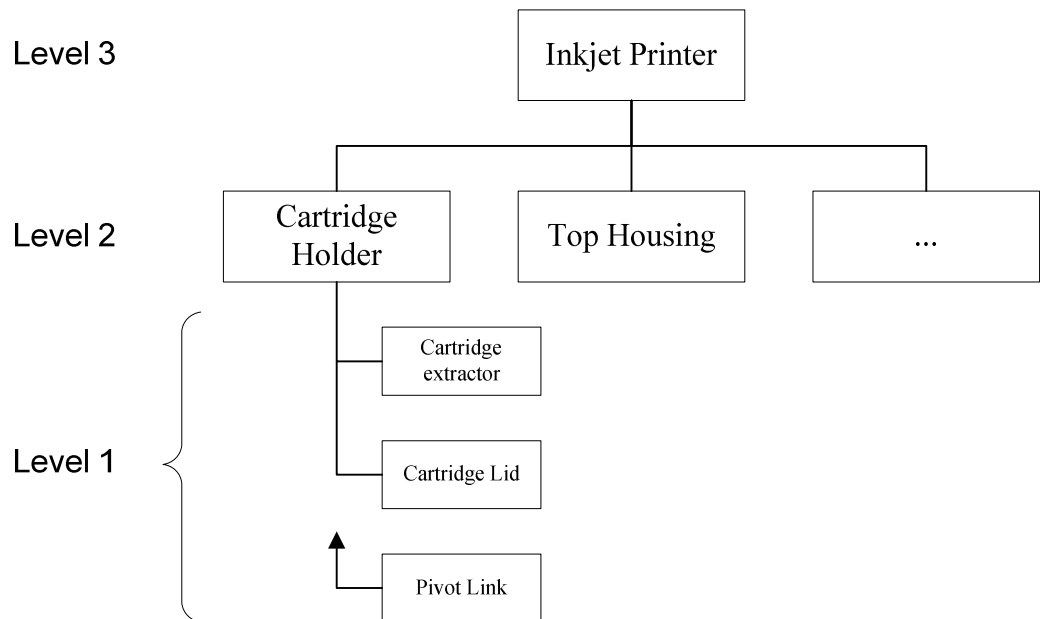


Figure 3.8 Creating new ontology in accordance with the object's complexity

3.3.3 Ontology Retrieval

Suppose the ontology set O holds a number of ontologies that contain the class of subject j and x_j be the item in j that the user would like to retrieve from O . Since the users may have created their own ontologies on the specific subject, therefore the algorithm will return ontologies of synonymous concepts by taking into account of attributes and structural equivalencies. The following rule is defined for retrieving ontology of relevant subject requested by the user from the ontology set. The algorithm for ontology retrieval is shown in Figure 3.9.

$$\forall o \in O \quad c(x) \equiv c(o) \rightarrow \text{return}(o) \quad (3.10)$$

```

begin
  /* Search for ontologies which contain subject that is equivalent to  $k$ , then they
    will be stored in a list  $L$  and returned*/
  for  $i = 1$  to  $m$ 
    if  $j \equiv k$  then appends  $o_i$  to  $L$ 
    end-if
  end-for

  /* Return  $L$  if it is not an empty set */
  if  $L \neq \emptyset$  then
    return  $L$ 
  end-if
end

```

Figure 3.9 Algorithm for ontology retrieval

In this chapter, an ontology-based data model is described in first order logic. The relationship of object and product database is defined through the two categories: *part* and *assembly*. The functions of the ontology-enabled PDM system provide users a practical way to access the synonymous information for solving the problem they are currently facing. By organising objects into

categories, users can infer their compositions from the perceived properties of the objects, and then use category information to make predictions about the objects. For example, the mass of a composite object is the sum of the masses of the parts. An implementation of the system for environmental compliance analysis is described in chapter 4. A model makes use of collaborative product design and manufacturing information management as the basis for environmental product development to analyse the content of hazardous materials in a product will be discussed.

Chapter 4

Ontology-based Environmental Compliance Management System

Apart from increasing pressure of shortening the time for product design and manufacturing, new laws are now forcing manufacturers to remove lead and other hazardous substances from their equipments, and to take responsibility for the eventual recycling of their products. Regulations on materials used in the products and on the influence of using the products on the environment have now become stringent and expect to be more restrictive. However, complex products require multidisciplinary design teams to master the design and to comply with safety and environmental regulations. One way to cope with these is to adopt a product development system that provides guidance on environmental issues.

4.1 Background on Environmental Compliance

Environmental damage caused by human drew attention to the impact of chemicals on the environment. People are encouraged to carefully dispose of unusable goods according to the type of material in early days, so that the potentially useful materials can be recycled. However, there are some goods that are difficult to be recycled; they have to be disposed of by other means like landfill and incineration. These methods may be environmental damaging and unsustainable. Furthermore, non-renewable resources are still being depleted and environmental pollution is increasing. Thus, many developed countries have progressed an extra step further on their environment protection policies, the European Union (EU) has implemented two new environmental directives: RoHS (Restriction of the use of certain Hazardous Substances in electrical and electronic equipment) and WEEE (Waste Electrical and Electronic Equipment). The RoHS directive took effect on 1st July 2006, which restricts the use of six hazardous materials in the manufacture of various types of electronic and

electrical equipment. It is closely linked with WEEE, whose purpose is to improve the reuse, recycling, and recovery in order to reduce the amount of disposal equipment and the contents going to landfill. For RoHS in EU, it requires that everything that can be identified as a homogeneous material must meet the content limit. The regulatory process is becoming more stringent, failing to comply with the regulations means that the products are banned. The manufacturers would not only suffer loss from not fulfilling the order but also ruins the company's reputation. The concern is further complicated when different countries have their own standards on the amount of substances presented in each product [Bergeson 2006].

In the past, companies are only required to develop and produce goods and services that are of consistently high quality, having shorter lead times and less expensive. Environmental issues have become important particularly in product development [Partidario & Vergragt 2002], environmental protection policies imposed by different countries make product development a very difficult and complicated task [Fawzi 2007]. Firms are now developing environmental policies for their operating facilities, services, and supply chain partners while trying to maintain consistency with new regulations. Many enterprises manufacture their products using materials and parts procured from various vendors. However, product development activities often exceed the boundary of one firm, the availability of information and resources are usually very low. Meanwhile, these companies often have difficulties in identifying whether all of the materials used are conforming to regulations. Consequently, they generally have difficulties in handling environmental issues with their production activities [Leistner 1999].

4.2 Environmental Compliance Management System

This section will begin by defining environmental management, after which the suitability of PDM system as the basis of environmental management system will be discussed. Environmental management is defined as encompassing all efforts to minimize the negative environmental impact of the firm's products

throughout their life cycle [Sayre 1996]. An environmental management system prevents adverse environmental effects and improves environmental performance by institutionalizing various environmental programs and practices such as initiating environment-related performance measures and developing green technologies, processes, and products.

Many firms realise the necessity to incorporate their supply chains with environmental compliance management (ECM). For example, in order for a firm to respond to customers' needs and to ensure its approach to the market in accordance with the regulations for hazardous materials, e.g. WEEE & RoHS Directives, it has to understand the environmental impacts of the parts and components supplied from its suppliers. In addition, the regulations for product take-back require it to expand its environmental responsibility to the entire life cycle of products. To respond to these requirements, firms have to incorporate an ECM system with their product design process and supply chain management.

4.2.1 Information Management Tools

To achieve the above improvements requires dealing with different enterprise functions and information sources. There are a number of commercial information management applications available in the market, such as:

- Product Data Management systems (e.g. PDMWorks [SolidWorks 2004], SmarTeam [IBM Corporation 2007], etc.) will keep track the data and information required to design products. PDM is used to work with electronic documents including CAD drawing, BOM, and product configuration.
- Enterprise Resource Planning systems (e.g. SAP [Keller & Teufel 1998]) will centralise all data and processes of an organization into one single database.

- Life Cycle Assessment (e.g. SimaPro 7 [PRé Consultants 2007], GaBi 4 [PE International 2007], DFE [Boothroyd Dewhurst Inc. 2007], etc.) is a tool that examines every stage of the life of products, including the production phase, distribution, use, and final disposal of the product.
- Compliance Management Applications (e.g. Materials Compliance Central [Enovia MatrixOne 2006], Compliance Management [Corporation 2007b]) will provide companies the ability to verify the compliance of a product by verifying the information of its material contents and to identify the non-compliant parts that are used.

These state-of-the-art software applications offer only partial responses to the needs of environmental compliance. They are specific tools or technology applications that optimize only parts of the product development process. Moreover, they are enterprise-centric information systems that require long time to set up and are not designed to provide a communication infrastructure for the whole project network. These inadequacies discourage establishments of virtual enterprise to work on ad-hoc developments of environmental compliance products.

An innovative methodology is thus proposed such that it enables different companies to incorporate their existing applications and decision-support functions into a web-based environment. The system can provide a distributed environment with enough flexibility to companies who have limited resources to form a working platform for developing environmental compliant products.

4.2.2 Architecture of PDM System

Like many other multi-client applications, PDM follows the three-tier architecture. Software development has been evolving in the last two decades; three-tier architecture divides an application into three distinct software agent. Multi-tier architecture is an open, distributed approach that separates the client into two parts – user interface, logic processing and the database, the overview

of a three-tiered application is shown in Figure 4.1. Other than the advantage of storing data remotely from users, the three-tier architecture is intended to allow any of the three tiers to be upgraded or replaced independently as requirements or technology change. A classic example of the three-tier architecture is the World Wide Web (WWW), where web browsers form the client tier, the database server forms the third tier, and the TCP/IP serves as the second tier. Given that PDM systems are developed based on the three-tiered distributed architecture, there are a number of similarities between web technology and PDM methodology, in terms of architecture and conceptual module.

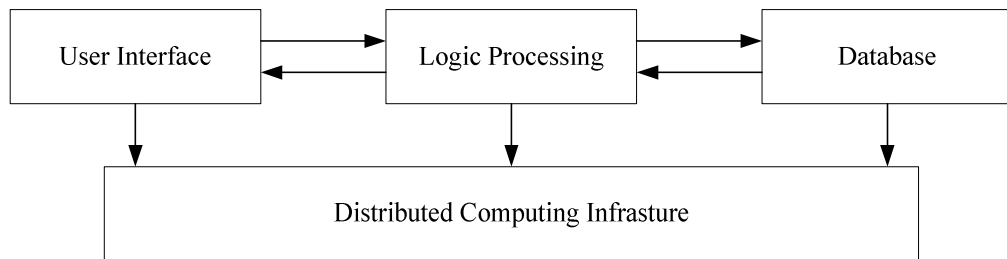


Figure 4.1 Overview of the three-tier architecture

The architecture of a PDM system can be divided into three tiers: the first tier is the user interface of the system, the second tier is the PDM logic server, and the third tier is the PDM database or repository as shown in Table 4.1. The table also shows that PDM methodology resembles web technology in the sense of their structures. The internet has brought the world a tremendous influence in communication. The use of web technology with PDM systems can create a network between enterprises and enhance departmental collaboration. The complement of web technology to PDM systems can create an affordable ECM system and helps companies to overcome the obstacles existed in environmental complied supply chain management.

	Web technology	PDM system
1 st tier	Web browser	PDM user interface
2 nd tier	Web / Application server	PDM logic server
3 rd tier	Web distributed database	PDM database / repository

Table 4.1 Structure of web technology and PDM system

4.3 The Model of Ontology-Enabled ECM System

An ECM system can be created by integrating web-technology with ontology-enabled PDM systems. This system possesses all PDM functions that allow a company to manage its product data locally. In addition, the system can also manage remote product data from its suppliers implicitly provided that a channel for data transfer is available to connect the management systems of the involved companies in the supply chain by the internet.

4.3.1 Structure of the System

The model of the ECM is designed to provide a foundation for companies to use existing PDM system to facilitate the implementation of ECM. Its structure is illustrated in Figure 4.2 and descriptions of the stages involved are presented.

1. Product design – When a product design team starts designing a product, the Compliance Analysis Module will create a new ontology for the new project in project database to categorize the meta-data of all the relevant product and document.
2. PDM system – It is the central unit of the ECM system for all the parties involved in the production process to communicate and to work on the project. The description of the module will be explained in more details in later sections.

3. Production planning – Given that the product design passes the relevant compliance, the ECM system will notify the production planning unit to assess the operational ability of the existing production system on manufacturing this new design and to determine the amount of materials required and the production schedule.
4. Procurement – After production planning has been completed, procurement will start sourcing the materials. Information of the materials selected is retrieved from the PDM system and stored into the project database through the analysis module. Purchase Orders will be sent to the suppliers once the materials satisfy the compliance standards.
5. Suppliers – Sometimes information of materials and parts are not available in the PDM system, the module will prompt the suppliers to provide the missing information. In case the suppliers are not able to supply any of the materials, they will reply the company using the system. The module will search through the PDM ontology repository for alternatives and suggest them to the design team.

The model has been designed to adapt the product development process in a project-oriented nature. A project database is established in the initiation of a new product development project. This gives a greater flexibility to the product developers as each project has its own complexity of technical objectives, since many management systems often provide a general management model, which may fail to capture the complicate relationship between each of the parties involved.

This model provides a unique entry point to all activities and data associated to the project by using the PDM system to control the access to the project database. This will allow independent data maintenance to the product development company and its suppliers. Through the use of web technology, the distributed environment facilitates remote site design teams to be able to update product data from any location directly accessing the project database.

Also, companies will no longer be tied up to any information management technology nor change their current streamlined practice for the sake of fitting to a particular commercial tool.

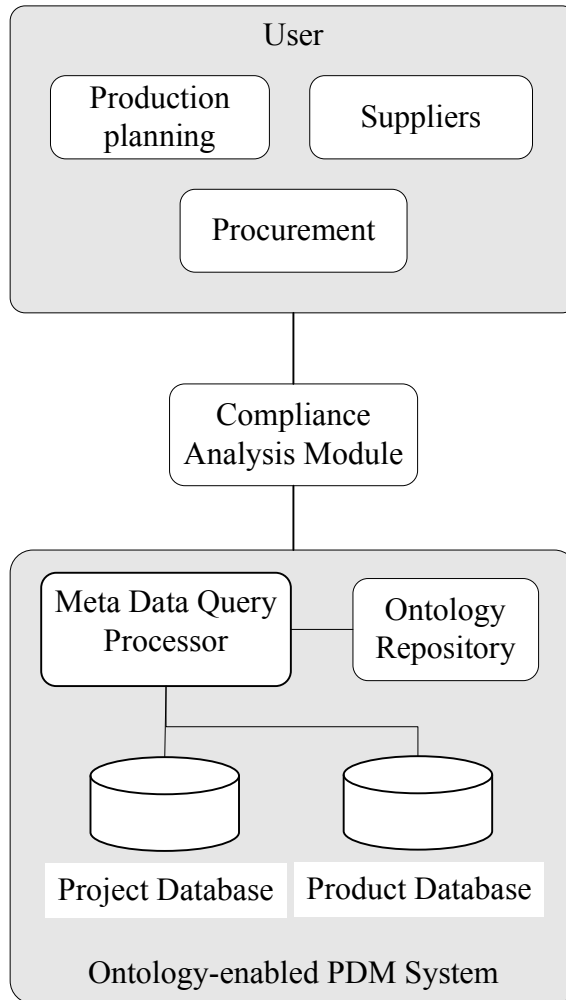


Figure 4.2 Model of environmental compliance management system

4.3.2 The Compliance Analysis Module

In order to provide a collaborative working environment for users from all related companies in the supply chain, a web-enabled PDM system is utilised to manage all product-related data and to provide data retrieval for product design and production. To analyse whether a product is compliant to a set of regulations, users must start the process by using the unique interfaces of the

Compliance Analysis Module for security and data integrity reasons. The procedure of analysing the compliancy of a design shown in Figure 4.3 is described as follows:

1. Data acquisition - In the beginning of the compliance analysis, the user selects the product need to be analysed, the module will then retrieve all the relevant information from the PDM system database.
2. Supplementary data - Depending on the completeness of the data and the kind of analysis to be performed on the product, the module will prompt the user to input the missing information or search the PDM system database of the company who supplied the part in question.
3. Compliance analysis – The data of the selected product will be verified against the corresponding regulations. Any violation of the regulation will be identified and necessary remedial action will be suggested to the user. At this stage, the user can configure the application to display the result according to region specific regulations to where the product is exported.
4. Preliminary report – Checklist of the analysis is drafted in the report. Marginal passes of any limits will be highlighted. All violations will be recorded and possible remedial actions will be listed in the preliminary report.
5. Re-analysis – Any non-compliance parts will be identified. The module will automatically search for alternatives in the PDM system. The user can make a component change based on the recommendation and the new design will be analysed again.
6. Format report – The preliminary report is formatted to the standard format required by the authority that established the regulations. The final report is then stored to the PDM system for the next analysis in case of alternations are made to the product. Instead of performing a

full analysis, a partial analysis can be run by detecting the changes and evaluate their effects accordingly.

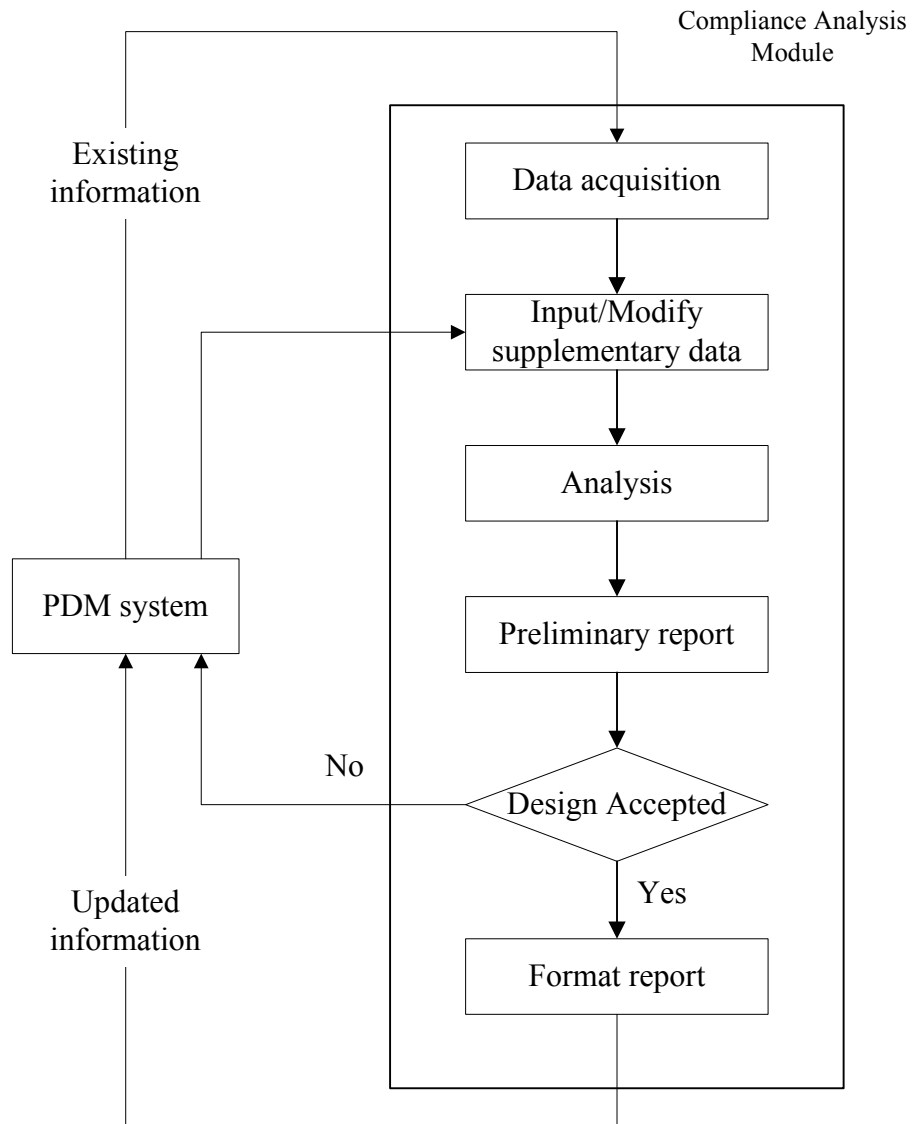


Figure 4.3 Procedure of compliance analysis

4.4 Implementation of ECM System

This section presents the implementation of the model using a PDM system and shows how the environmental management system helps a company to analyse the compliance of a product in the WEEE directive in a timely manner.

4.4.1 Software for Implementation

The system is designed and implemented using normal client/server architecture with several free and some relatively inexpensive commercial tools.

PDMWorks – In this research, the PDM system - PDMWorks Enterprise has been chosen for developing the ECM system. It is the latest product data management (PDM) software from SolidWorks® to help engineers and product managers work more efficiently in teams while automating workflow. The reason of choosing this PDM system is that the software supplies comprehensive controls to help the design team avoid the possibility of overwriting files or making other errors than could add time and cost to their schedule. It helps organizations better control each design project while streamlining development.

PHP - The interface of the system is developed using PHP. It is a widely-used general-purpose scripting language that is especially suited for Web development and can be embedded into HTML. PHP can be deployed on most web servers and on almost every OS platform free of charge.

MySQL – It is the database system used in the implementation. MySQL runs on many OS platforms including Linux, Windows, etc., which provides a great flexibility in configuration.

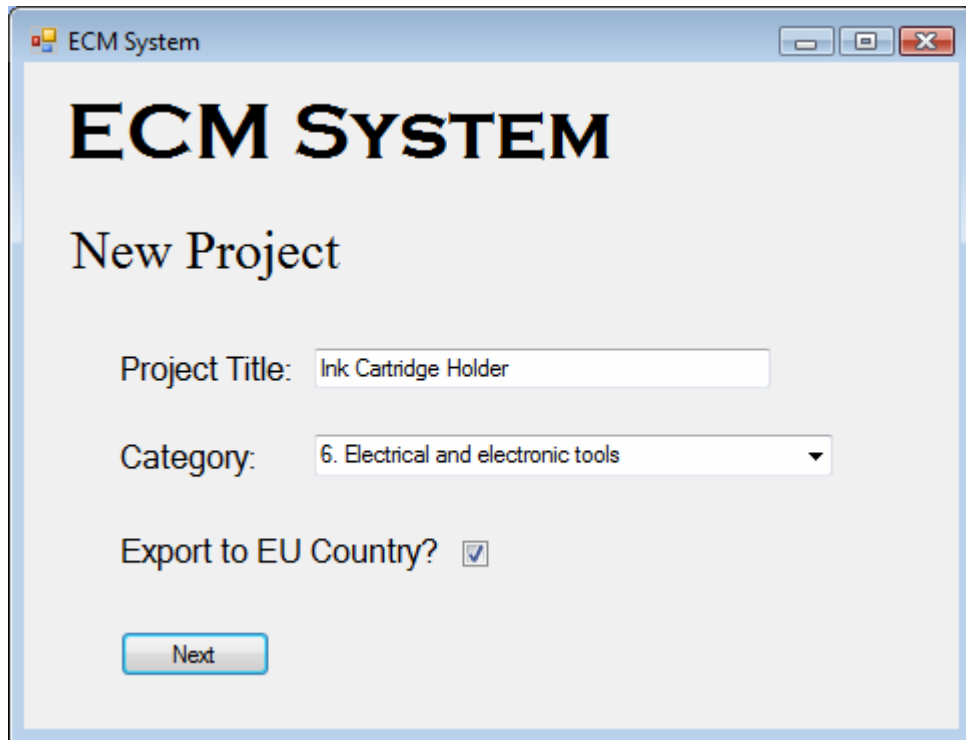
4.4.2 The Compliance Analysis Module – ECM System

The Compliance Analysis Module is the second tier of a multi-tier architecture system that is responsible for processing users' requests. An application called ECM System is the process-tier in the PDM system which aims to provide the effective use of the network infrastructure, to facilitate the cooperation between the users, to support workflow automation, to efficiently manage document retrieval, and to provide the user a suitable interface in order. ECM System is

developed to analyse the compliancy of products efficiently based on the EMS model described in the previous section. In general, ECM System provides a co-operating channel with PDM software through importing CSV (Comma Separated Variables) report and exporting drawing file. The CSV report is generated by the PDM software from the bill of materials of the project. In case of any missing of necessary data, the user can input those missing data into ECM System by the preset form. It will analyse the data and generate a compliance report when the data acquisition is completed. The user can decide to export the report or modify the current data depending on the report. The exported report is in engineering drawing file format for better integration with PDM software, and it could be previewed in the PDM software directly through eDrawing plug-in. More information on the plug-in is listed in Appendix A. At the beginning of the analysis process, the user can select to create a new project or open an existing project for modification.

4.4.3 Illustrative Example

A case study is presented in this section to illustrate the compliance analysis using ECM System with PDMWorks. In the process of designing a new product, the user is prompted to provide some basic project information for compliance analysis that includes the project title, the appropriate WEEE product category and the product's lifetime. The start-up page of ECM System asks a user to enter the name of the project and select the category to which the product belongs is shown in Figure 4.4.



The screenshot shows a web application window titled "ECM System". The main heading is "ECM SYSTEM" in a large, bold, black font. Below it is the sub-heading "New Project". There are three input fields: "Project Title" with the text "Ink Cartridge Holder", "Category" with a dropdown menu showing "6. Electrical and electronic tools", and "Export to EU Country?" with a checked checkbox. A "Next" button is at the bottom.

Figure 4.4 Start up page of ECM System

The new project is created and written into the project database. After entering the general information using the predefined template, ECM System will prompt the user to provide the data of all parts in use in the project by selecting the data from the product database in the PDM system. In case of using new part data, they must be registered before use through the meta-data processor. A new ontology is created for the new design by analysing the relationships between each of the parts and assemblies in the product. Like PDMWorks, the part data can be exported by the reporting function to make a CSV report file. The BOM of the product will be displayed to the user for inspection in case there is any data missing, as shown in Figure 4.5.

The screenshot shows a software window titled "ECM System" with a "BILL OF MATERIAL" header. Below the header, the project name "Project: Ink Cartridge Holder" is displayed. The main content is a table with two columns: "Part" and "Material". Each row represents a component of the product, listing the part name, the material selected from a dropdown menu, the quantity, weight, unit, and a checked "Recyclable?" checkbox. At the bottom of the window, there are "Save" and "Cancel" buttons.

Part	Material
Cartridge lid	SPI#2 HDPE - High Density Polythylene Qty 1 Wgt 0.06 kg Recyclable? <input checked="" type="checkbox"/>
Cartridge back board	SPI#2 HDPE - High Density Polythylene Qty 1 Wgt 0.03 kg Recyclable? <input checked="" type="checkbox"/>
Cartridge extractor	SPI#2 HDPE - High Density Polythylene Qty 1 Wgt 0.06 kg Recyclable? <input checked="" type="checkbox"/>
Cartridge latch	SPI#2 HDPE - High Density Polythylene Qty 1 Wgt 0.02 kg Recyclable? <input checked="" type="checkbox"/>
Cartridge holder	SPI#2 HDPE - High Density Polythylene Qty 1 Wgt 0.1 kg Recyclable? <input checked="" type="checkbox"/>
Ink Carriage	SPI#2 HDPE - High Density Polythylene Qty 1 Wgt 0.1 kg Recyclable? <input checked="" type="checkbox"/>
Link Support	SPI#2 HDPE - High Density Polythylene Qty 1 Wgt 0.02 kg Recyclable? <input checked="" type="checkbox"/>
Pivot link	SPI#2 HDPE - High Density Polythylene Qty 1 Wgt 0.06 kg Recyclable? <input checked="" type="checkbox"/>

Figure 4.5 BOM display of the product being analysed

Data in CSV files will be stored into the part database of ECM System directly.

The data structure of the part database is shown in Figure 4.6.

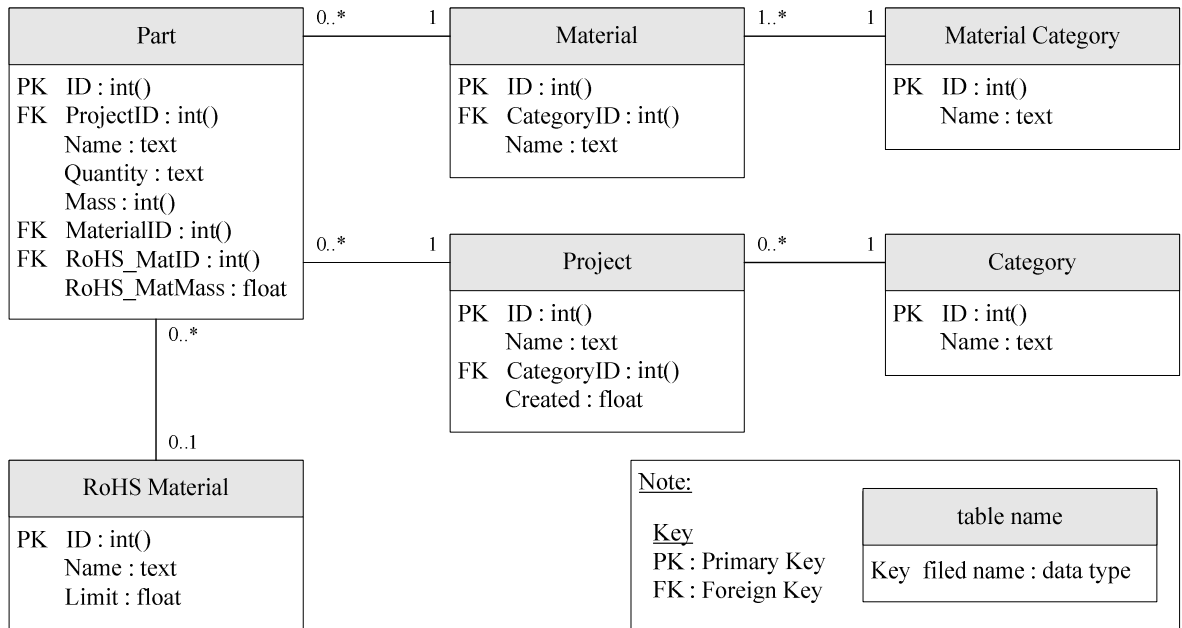


Figure 4.6 Data schema of part database in ECM System

After importing data from the PDM system, ECM System checks the availability of data required for compliance analysis. In case of any data missing, ECM System will locate the data using the information of supplier of the specific parts stored in the PDM system. It will automatically connect the PDM system with the supplier’s system and retrieve the data if both companies are committed to work collaboratively. Conversely, ECM System will alert the user that the analysis is performed upon incomplete information and the result may vary from the inclusion of the missing data.

Once the compliance analysis of the product is completed, a compliance report will be generated. The user can now view the result of the analysis that shows the level of reuse, recycling, and recovery. The grading of the compliance criteria is listed for each part of the product and the compliant status of the product is also enclosed. An example of a compliance analysis report is shown in Figure 4.7 and 4.8.

Ignore	Part	Recyclable	Recovery
<input type="checkbox"/>	Cartridge lid	<input checked="" type="radio"/> 13.33%	<input type="radio"/> <input type="text"/> %
<input type="checkbox"/>	Cartridge back board	<input checked="" type="radio"/> 6.67%	<input type="radio"/> <input type="text"/> %
<input type="checkbox"/>	Cartridge extractor	<input checked="" type="radio"/> 13.33%	<input type="radio"/> <input type="text"/> %
<input type="checkbox"/>	Cartridge latch	<input checked="" type="radio"/> 4.44%	<input type="radio"/> <input type="text"/> %
<input type="checkbox"/>	Cartridge holder	<input checked="" type="radio"/> 22.22%	<input type="radio"/> <input type="text"/> %
<input type="checkbox"/>	Ink Carriage	<input checked="" type="radio"/> 22.22%	<input type="radio"/> <input type="text"/> %
<input type="checkbox"/>	Link Support	<input checked="" type="radio"/> 4.44%	<input type="radio"/> <input type="text"/> %
<input type="checkbox"/>	Pivot link	<input checked="" type="radio"/> 13.33%	<input type="radio"/> <input type="text"/> %

Figure 4.7 Analysis on the grading of parts

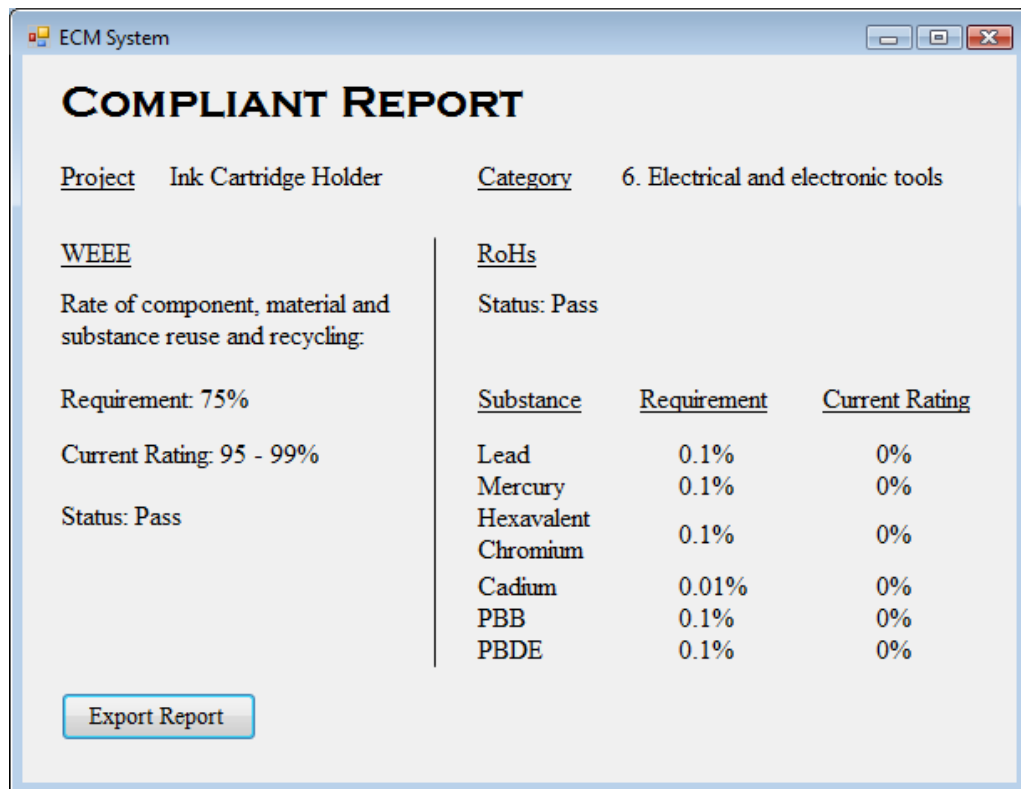


Figure 4.8 Report showing the compliant status of the product

The report can be previewed immediately and the user can choose to modify the part data or export the report. The report is formatted as a drawing file for performing a better integration with the PDM software. The analysis result can be exported as DXF format for PDMWorks import. DXF format provides a simple interface, easy to read, generates clean code, and has a very small footprint in terms of RAM and disk space requirements. Figure 4.9 is the report in DXF format. The report shows that the product being examined passes the WEEE directive, but fails to comply with the RoHS directive. The content of each of the six restricted substances is listed in the report, ECM System identified the lead content of the product exceeds the limit. The parts that contain the concerning substance are listed in the descending order of the content.

The screenshot shows a software window titled 'eDrawings - [Sample.dxf]'. The main content area displays a 'COMPLIANT REPORT' for the project 'Ink Cartridge Holder' under category '6. Electrical and electronic tools'. The report is divided into two main sections: WEEE and RoHs. The WEEE section includes 'Rate of component, material and substance reuse and recycling: Requirement: 75%', 'Current Rating: 95 - 99%', and 'Status: Pass'. The RoHs section includes 'Status: Pass' and a table of substances with their requirements and current ratings.

<u>COMPLIANT REPORT</u>																						
<u>Project</u> Ink Cartridge Holder	<u>Category</u> 6. Electrical and electronic tools																					
<u>WEEE</u>	<u>RoHs</u>																					
Rate of component, material and substance reuse and recycling:	Status: Pass																					
Requirement: 75%																						
Current Rating: 95 - 99%																						
Status: Pass																						
	<table border="1"> <thead> <tr> <th><u>Substance</u></th> <th><u>Requirement</u></th> <th><u>Current Rating</u></th> </tr> </thead> <tbody> <tr> <td>Lead</td> <td>0.1%</td> <td>0%</td> </tr> <tr> <td>Mercury</td> <td>0.1%</td> <td>0%</td> </tr> <tr> <td>Hexavalent Chromium</td> <td>0.1%</td> <td>0%</td> </tr> <tr> <td>Cadmium</td> <td>0.01%</td> <td>0%</td> </tr> <tr> <td>PBB</td> <td>0.1%</td> <td>0%</td> </tr> <tr> <td>PBDE</td> <td>0.1%</td> <td>0%</td> </tr> </tbody> </table>	<u>Substance</u>	<u>Requirement</u>	<u>Current Rating</u>	Lead	0.1%	0%	Mercury	0.1%	0%	Hexavalent Chromium	0.1%	0%	Cadmium	0.01%	0%	PBB	0.1%	0%	PBDE	0.1%	0%
<u>Substance</u>	<u>Requirement</u>	<u>Current Rating</u>																				
Lead	0.1%	0%																				
Mercury	0.1%	0%																				
Hexavalent Chromium	0.1%	0%																				
Cadmium	0.01%	0%																				
PBB	0.1%	0%																				
PBDE	0.1%	0%																				

Figure 4.9 ECM System created analysis report in DXF format

An environmental compliance management system developed upon an ontology-enabled PDM system and web technology is discussed in this chapter. The model includes a compliance analysis module that retrieves product data from a PDM system to analyse the content of hazardous materials in a product. Information regarding the design of interest can be compiled by checking all its assemblies and parts. The ontological organisation of the components facilitates the verification of the content of materials and parts used within the product, thus the losses incurred from violating environmental regulations can be prevented. Chapter 5 will describe the functions of PDM system using UML sequence diagrams and formal notations. Common concurrency problems in a distributed data management environment and the difficulties of developing the specification of PDM system will also be discussed.

Chapter 5

Representation of PDM Functions in UML Sequence Diagram

5.1 PDM User Functions

PDM provides benefits in every area of product design and development. Almost everyone in an organisation can gain an advantage through the use of this technology. A PDM system possesses a number of functions to support any particular type of product development. This chapter provides a summary of the functionality of PDM systems. CIMdata [CIMdata 2005] divided the basic functionality of PDM systems into two main categories: user functions and utility functions. The user functions allow users to store, retrieve, and manage the data in a PDM system and the database consistency is depending on the order of the executions of these functions. A PDM system commonly involves two kinds of data storage. Figure 5.1 summarises the data flow in a PDM system under the operations of the basic function of document management. An electronic vault is a repository to store all kinds of product information that are used in a collaborative work environment. Only authorized users of the PDM system can gain access to the vault, it is necessary for the users to check out the required files from the vault before any file operation is processed. Once the user has ownership of the files, they are copied to the user's local workspace, such that the user can work more efficiently with the files locally. The basic PDM system functions are the common operations to work with data objects in a PDM system. These functions are categorized into two main groups: user functions and intrinsic functions. The six user functions allow users to store, retrieve, and manage the data in a PDM system. User functions are blocks of actions to be performed upon evocation. Intrinsic functions are the actions that respond to the user functions and cannot be evoked by users directly. Descriptions of the basic functions of PDM system are covered in the following sections.

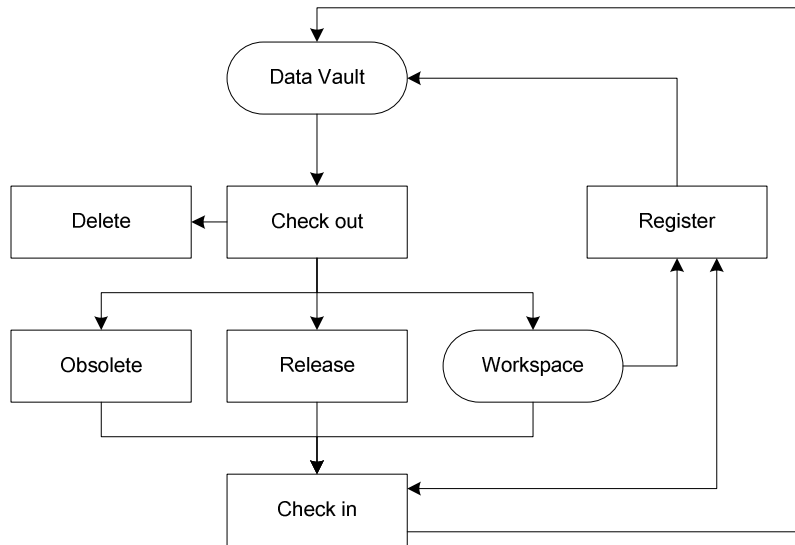


Figure 5.1 Flow of data under the operation of various PDM functions

5.1.1 Description of PDM System User Functions

PDM_Register(*d*) Upload a newly created data object to the PDM database if the object does not exist in the database.

PDM_CheckOut(*d*) Save the data object to the workspace from the PDM database if the object is available for specific tasks.

PDM_CheckIn(*d*) A previously checked-out object becomes available for other accesses. A new version of the data object will be uploaded if it has been modified.

PDM_Release(*d*) The data object can no longer be modified if it is released.

PDM_Obsolete(*d*) The obsoleted data object cannot be retrieved for any tasks.

PDM_Delete(*d*) The deleted data object is removed from the PDM database and no longer exist in the system.

5.1.2 Description of PDM System Intrinsic Functions

Exist(*data object, server*)

Return *true* if the data object exists in the PDM system by checking the metadata in a server, *false* otherwise. This function also sets the existence of the data object in the system.

Save(*origin, destination, data object*)

Save function makes a copy of the data object to other place. It takes three arguments, they are the system components in which the data object is residing, the locations that is going to store the object and the objects to be transferred in the respective order.

Available(*data object*)

Return *true* if the data object is available for a specific operation. The availability of an object is determined by the mode of lock that is applied by actions other users subjected to their operations. This function also sets the availability of the data object in the system.

Modifiable(*data object*)

Return *true* if the data object can be modified.

Erase(*data object*)

The data object is removed from the PDM system if it is in obsoleted state.

5.2 Use of Sequence Diagram

The proposed approach for integrating UML and a formal language does not regard all of the diagrams specified on UML specification. In this study, the UML sequence diagram is chosen as the visual modeling tool given that interactions among objects take place in a specified sequence, and the sequence takes time to go from beginning to end. First order logic (FOL) will be used to describe the specifications of these functions. The order of the sequence of actions can be clearly presented in the sequence diagrams and that facilitates the process of developing the specification of the system with formal notations in which system developers can verify its correctness properties.

Summary of the user functions are illustrated using the UML sequence diagrams. Sequence diagrams are read from the top to the bottom and left to right. The construct of the FOL formulas that based on the order of the function predicates are asserted. A function predicate can be asserted to hold at any state of the lifespan of an object. Unless function predicates are refuted, they will remain hold from the state at which functions are evoked. An action function is an expression of the executability of the specific function. For example, $PDM_Register(d)$ holds when the data object d can be registered to a PDM system. Likewise, property function reflects the particular property of the object, i.e. $Available(d) = T$ means that d is available for operations.

5.2.1 Registering data in PDM System

The process of registering a new data object saves a copy from the user's workspace to the data fault of the PDM system, which is shown in Figure 5.2. PDM server verifies the meta-data to determine whether it already exists in the database. The registered data object can then be accessed by other authorized users. If the data object is an existing data object, the server will inform the user to ascertain that the original would not be overwritten unintentionally.

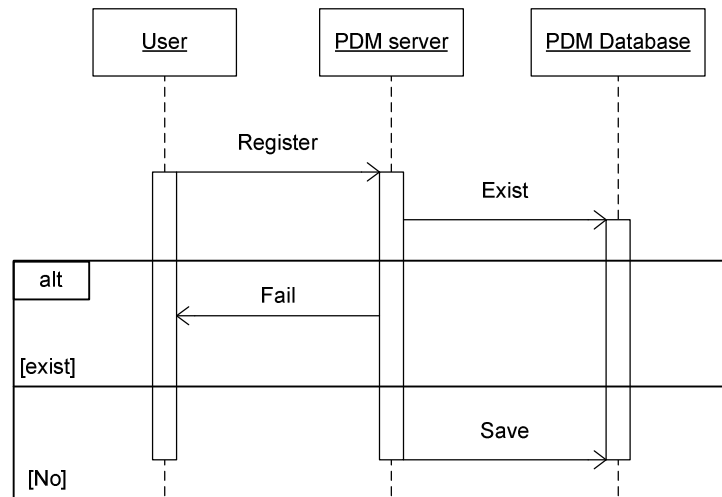


Figure 5.2 Sequence diagram for registering data

The following definition of the register function said that if the data object to be registered does not exist in the data vault by checking the metadata in the server, it will be saved to the data vault.

$$PDM_Register(d) \Leftrightarrow [\neg Exist(d, s) \rightarrow Save(s, db, d)] \quad (5.1)$$

5.2.2 Check-Out in PDM System

In many commercial PDM systems, user is only allowed to check out unoccupied documents, the following sequence diagram illustrates a standard check-out process. The system decides the check-out process based on whether the document is occupied or not. If it is being occupied, then the check out process fails. Otherwise, the system will lock the document in the data vault and grant the ownership of the document to the user. A PDM system with a multi-granularity locking mechanism has a different check out process. The check out request will only be declined if the data object is write-locked. In other cases, the system will lock the data object with a shared read lock and a read access is granted to the user. To cater for the transference in distributed environment, the data object is transferred to the PDM server to which the user issues the check out request and then saved to the user's workspace.

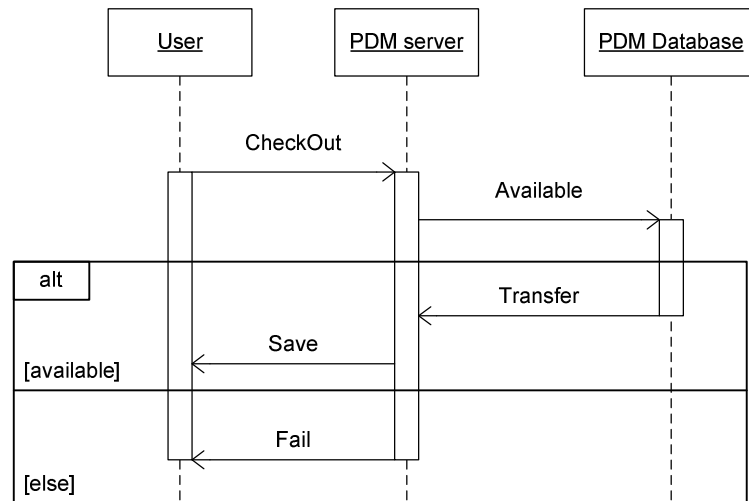


Figure 5.3 Sequence diagram for checking out data object

The check-out function is well defined if data object d is resided in the database db and the availability of d must be ascertained. The data object d is then transferred to the server s and saved to the workspace ws and d is no longer available for other users.

$$\begin{aligned}
 PDM_CheckOut(d) \Leftrightarrow [d \in db \mid Available(d) \rightarrow Save(s, ws, d) \\
 \Rightarrow Available(d) \rightarrow F]
 \end{aligned}
 \tag{5.2}$$

5.2.3 Check-In in PDM System

Figure 5.4 shows the process of saving a copy of the modified data object to the data vault of the system by using the check-in function. The process invokes the version control module to revise the attributes of the modified data object in the meta-database. Version control is an internal procedure for managing properties of data object in a PDM system. This creates or updates the meta-data, such as changes or version, for the data file according to the rules established by the system administrator. Having the right to override the rules of revision, the user can manipulate the revision to some extent.

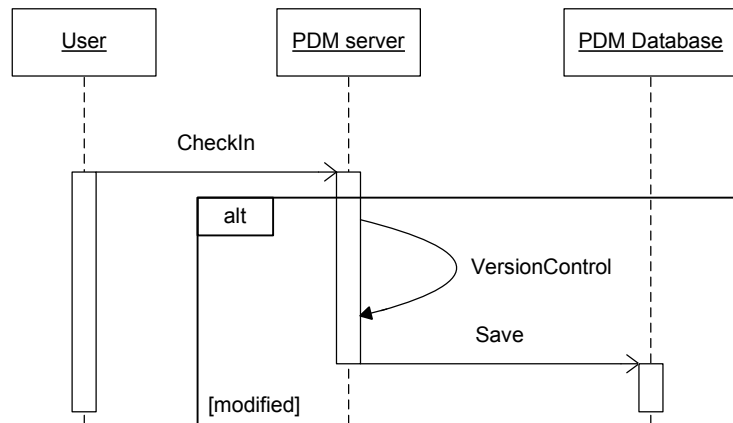


Figure 5.4 Sequence diagram for checking-in data object

The check-in function is defined as passing data object d' to the data vault from where it was drawn. This function has two formulas: either (5.3) or (5.4) will determine the validity of the process. In (5.3), the check-in object is the original one that has been checked-out, that is $d' = d$, then the data object becomes available again. In (5.4), the data object has been modified from $d \rightarrow d'$, the new version is checked-in to the database and become available for other users.

$$\begin{aligned}
 PDM_CheckIn(d') &\Leftrightarrow [d \in db \mid d' = d \wedge PDM_CheckOut(d) \\
 &\Rightarrow Available(d) \rightarrow T]
 \end{aligned}
 \tag{5.3}$$

$$\begin{aligned}
 PDM_CheckIn(d') &\Leftrightarrow [d \in db \mid d \rightarrow d' \wedge PDM_CheckOut(d) \\
 &\Rightarrow (Available(d) \rightarrow T \wedge Register(d'))]
 \end{aligned}
 \tag{5.4}$$

5.2.4 Release in PDM System

The release function restricts further modification to a data object. Figure 5.6 illustrates the process of the function in PDM system. The data objects cannot be modified anymore once it is finalised and approved by authorised users.

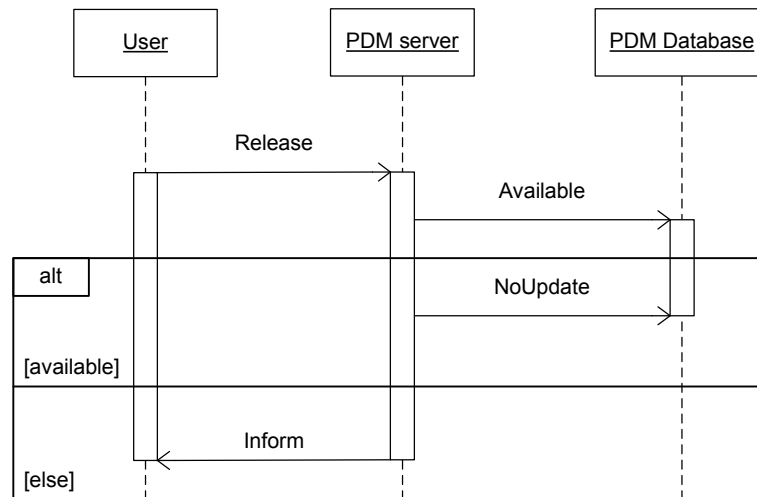


Figure 5.5 Sequence diagram for releasing data object

The formula of release function requires checking whether the data object d is available before further process. Once the data object is released, this object cannot be modified anymore.

$$PDM_Release(d) \Leftrightarrow [Available(d) \Rightarrow Modifiable(d) \rightarrow F] \quad (5.5)$$

5.2.5 Obsolescence in PDM System

The server checks whether there is any user accessing the data item on which the obsolete functions is acting. Once a data item is in an obsoleted state, all of its corresponding information cannot be used in any project starts thereafter. Occupy is an internal function to check the data object d is in-use or a component of any project in the system. If the inference of occupy is true, it means that the data object is either being used by someone or is part of other project, thus d cannot be obsoleted. Otherwise, if the process is successfully processed, then d becomes inaccessible.

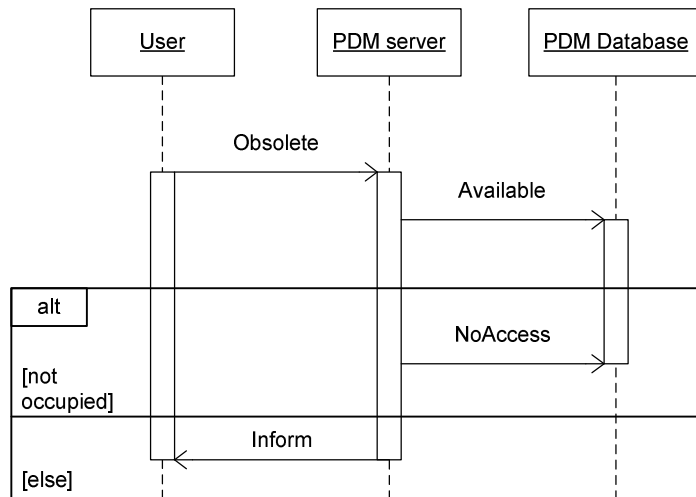


Figure 5.6 Sequence diagram for data object obsolescence

The formula of obsolescence ensures that the to-be-obsolete data object d is not included in any project and occupied by any user. Once d is obsolete, the data object in the PDM database is no longer accessible.

$$PDM_Obsolete(d) \Leftrightarrow [Available(d) \Rightarrow Available(d) \rightarrow F] \quad (5.6)$$

5.2.6 Deletion in PDM System

This function erases the chosen data item that must have been obsolete.

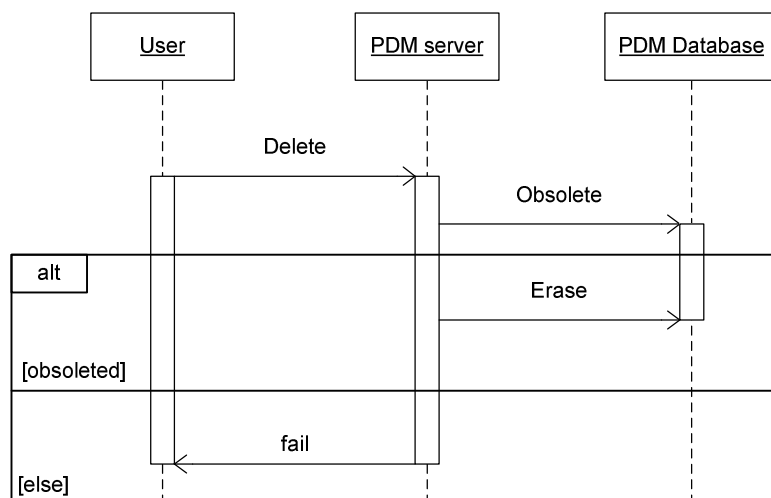


Figure 5.7 Sequence diagram for deleting data object

The following formula is the specification of *Delete* function. It defines that the data object d must be obsoleted. This implies that d is longer used in any project, then the server will invoke the *Erase* function to delete the data object, and therefore it is no longer an element of the database db .

$$PDM_Delete(d) \Leftrightarrow [Obsolete(d) \rightarrow Erase(d) \Rightarrow d \notin db] \quad (5.7)$$

5.3 DPDM User Functions

A Distributed Product Database Management (DPDM) system is one in which the database is spread among several sites and application programs. The DPDM operates more or less like the PDM system. DPDM system consists of all PDM functions in order to manage the product data at the local database. In addition, DPDM system must provide the data transfer between databases at different sites. The DPDM functions and their logical formulas are shown in this section. In the following illustrations, each of the PDM servers in the DPDM system holds a copy of the same meta-data. The advantage of administrating the meta-database is obvious: because each organisational unit within the enterprise will maintain data that is relevant to its own operations, it is often not necessary for each database to keep a copy of the actual data object, users from other sites may not require some of the data at all, thus unnecessary data flows between the servers are avoided. The distributed arrangement also improves the efficiency of processing by keeping the data close the place where it is most frequently used. Retrieving data from remote databases that possess the data when needed allows better data accessibility.

The group of function predicates used in constructing FOL formulas for PDM system are also used in developing those for DPDM system. Nonetheless, the function predicates take more arguments in order to handle the more complicated relationships between computers and servers in which the related actions take place. For example, the $PDM_CheckIn(d)$ function for PDM system checks in the data object d to the PDM database and the system has implemented only one centralised database to store the data. Although, DPDM system is composed of

several PDM servers and databases, it does not require users to explicitly specify the components of the system involved when they use these functions. Therefore, a user function in both PDM and DPDM system takes the same number of arguments. On the other hand, DPDM system needs to handle interactions between a number of system components, its intrinsic functions have to take extra arguments to show the components involved in an action. For instance, the intrinsic function for checking the availability of a data object in a DPDM system, *available(d, s_r)*, has an extra argument showing that the specific server is being checked.

5.3.1 Description of DPDM System User Functions

The following DPDM user functions perform the same actions of the respective PDM functions. However, the operations involved in each DPDM functions are slightly different and will be described in later section.

DPDM_Register(*d*)

DPDM_CheckOut(*d*)

DPDM_CheckIn(*d*)

DPDM_Release(*d*)

DPDM_Obsolete(*d*)

DPDM_Delete(*d*)

5.3.2 Description of DPDM System Intrinsic Functions

The following DPDM intrinsic functions take more arguments than respective functions in PDM system.

Available(*data object, server*)

Return *true* if the data object is available in particular servers for a specific operation. The availability of an object is determined by the mode of lock that is applied by actions other users subjected to their operations. This function also sets the availability of the data object in the servers of the DPDM system.

Modifiable(data object, server)

Return *true* if the data object can be modified in the server. This function also sets the whether the data object can be modified or not.

Erase(data object, server)

The data object is removed from the server in a DPDM system if it is in obsoleted state.

5.3.3 Registering data in DPDM system

The process of registering a new data object in DPDM system saves a copy from the user's workspace to the local PDM data fault, which is shown in Figure 5.8. The local PDM server verifies its meta-data to determine whether it already exists in the local database. If the data object being registered exists in the DPDM system, the system will inform the user to ascertain that the original would not be overwritten unintentionally. In a situation where the data object is new to the DPDM system, the data object is saved to the local database and the meta-data of the object is written to meta-database of all PDM servers in the DPDM system.

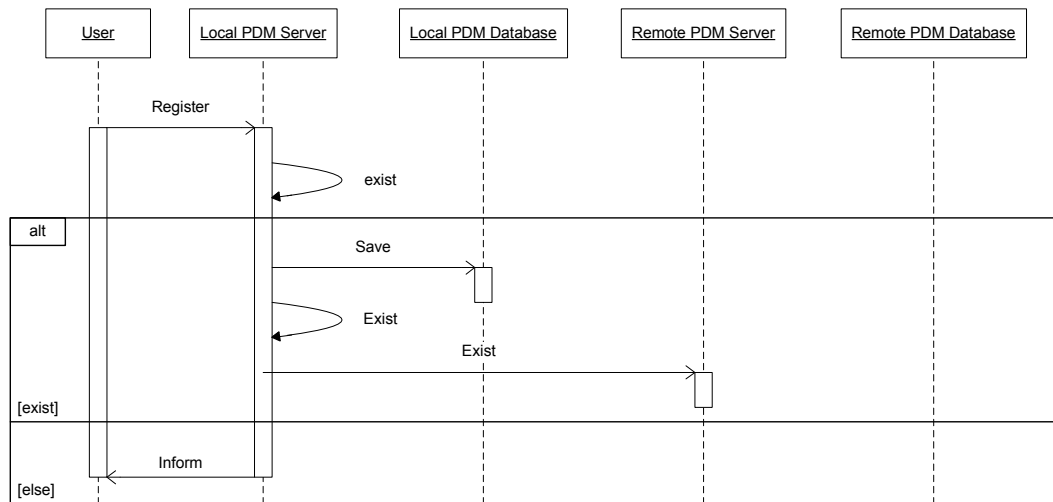


Figure 5.8 Sequence diagram for registering data in DPDM system

The following definition of the register function for DPDM system said that if the data object d to be registered does not exist in the local meta-database at the local server s_l , it will be saved to the local database db_l and the meta-data of d is written to the meta-database in the local server and in all remote servers of the DPDM system. The $\forall i Exist(d, s_i)$ function updates all servers s that d exists in the PDM system.

$$DPDM_Register(d) \Leftrightarrow [\neg Exist(d, s_l) \rightarrow (Save(s_l, db_l, d) \wedge \forall i Exist(d, s_i))] \quad (5.8)$$

5.3.4 Check-out in DPDM system

As in PDM system, the DPDM system decides the check-out process based on whether the requested data object is occupied or not. Figure 5.9 shows the process of check out function of DPDM system. If the requested data object is being occupied, then the check out process fails and the user is informed. Otherwise, the system will grant the ownership of the data object to the user by changing the status of the data object in all the meta-databases. If the data object resides in the local database, the local server will retrieve the data object from the database and then saves it to the user's workspace. On the other hand, to cater for the transference in distributed environment, if the requested data object is in one of

the remote databases, the local server will retrieve the data object from the remote database and saves it to the local database before transferring it to the user's workspace.

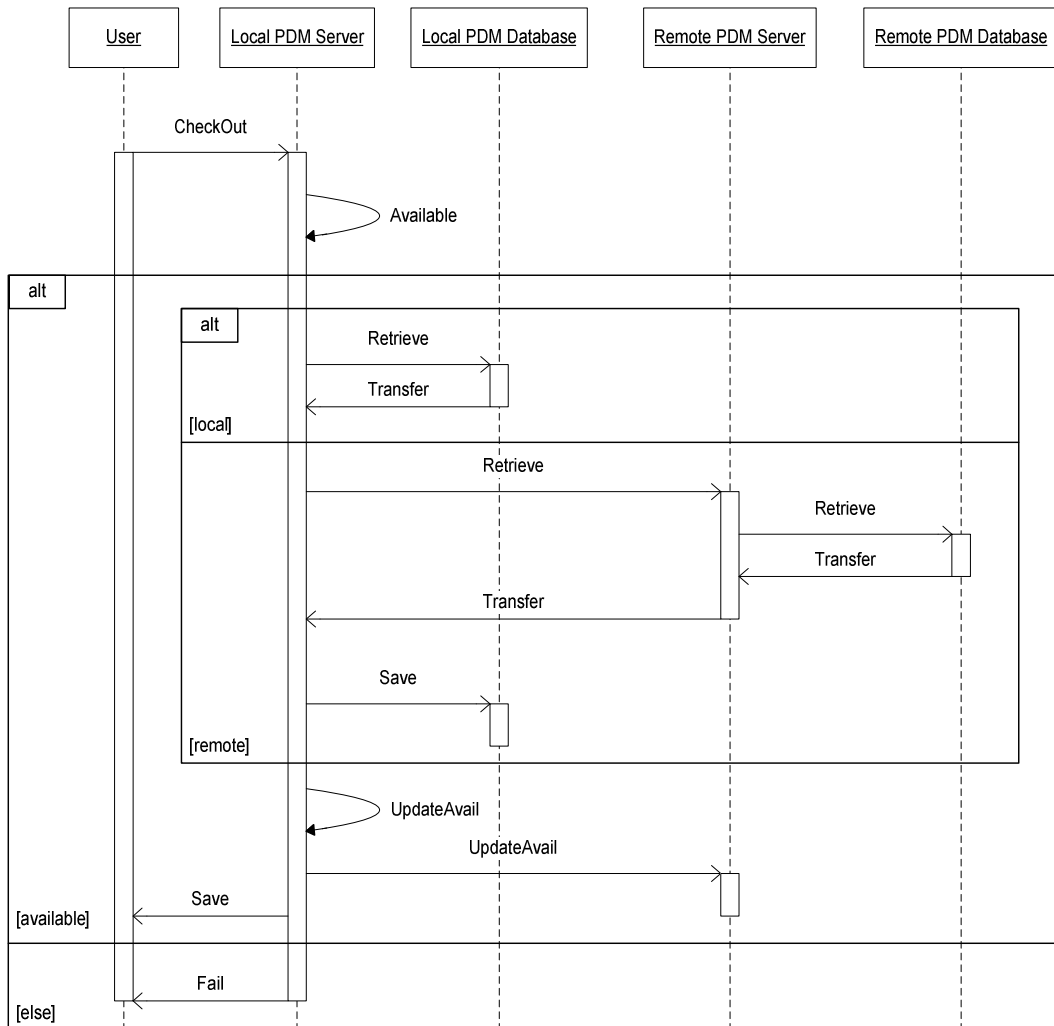


Figure 5.9 Sequence diagram for checking out data object in DPDM system

The check-out function is well defined if data object d is resided either in the local database db_l or in one of the remote database db_r . In the first scenario in which d is in the local database, the local PDM database passes the data object to the local server s_l first. In the later scenario in which the requested object is resided in a remote database, d will be saved to local database db_l from the remote database db_r . At the end of the check-out process for both of the scenarios, the availability

of d in the meta-database in all DPDM servers is changed to false and the data object is saved to the workspace ws .

$$DPDM_CheckOut(d) \Leftrightarrow [d \in db_l \mid \exists r (Available(d, s_l) \wedge (Exist(d, db_l) \vee Exist(d, db_r))) \rightarrow Save(db_r, db_l, d) \rightarrow (Save(s_l, ws, d) \wedge \forall i Available(d, s_i) \rightarrow F)] \quad (5.9)$$

5.3.5 Check-In in DPDM System

Figure 5.10 shows the process of saving a copy of the modified data object to the data vault of the system by using the check in function of DPDM system. The function is similar to the PDM system's version with some additional processes in updating the system record. The process invokes the version control module to revise the attributes of the modified object in the meta-database and to update the meta-database of the remote PDM servers.

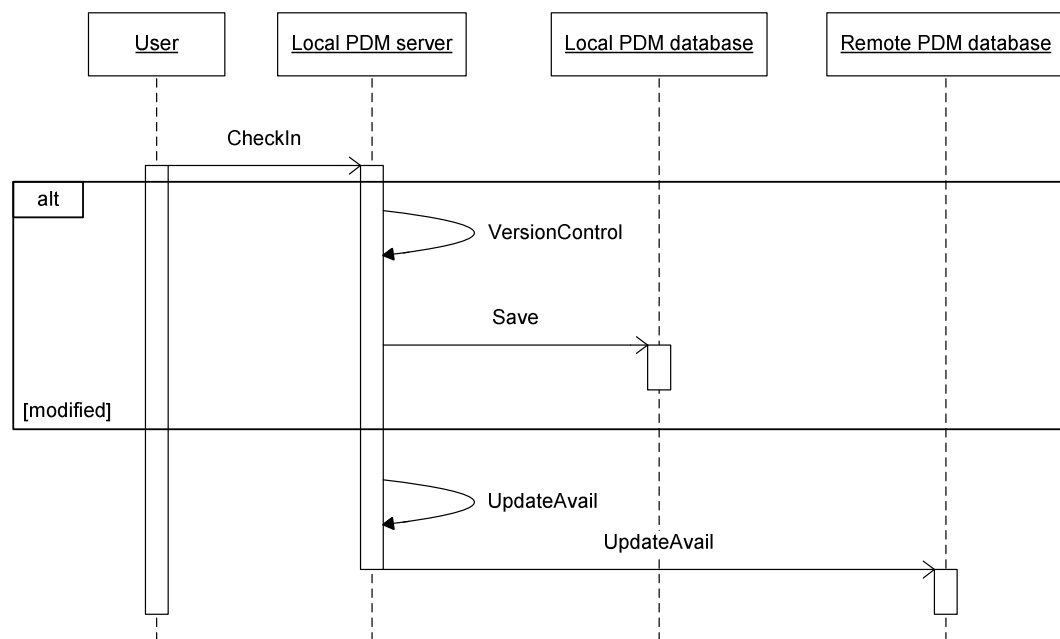


Figure 5.10 Sequence diagram for checking-in objects in DPDM system

In DPDM system, the check-in function is defined as passing data object d' to the local data vault. This function has two formulas: either (5.10) or (5.11) will

determine the validity of the process. In (5.10), the check-in object is the original one that has been checked-out, that is $d' = d$, and becomes available again. In (5.11), the data object has been modified from $d \rightarrow d'$, the old version of the data object is checked-in to the local database and become available for other users and the new version is registered to the DPDM system by the *Register* function. Both formulas ensure that the process updates the availability of d is updated to true in meta-database of all remote servers.

$$\begin{aligned} DPDM_CheckIn(d') &\Leftrightarrow [d \in db_i \mid d' = d \wedge DPDM_CheckOut(d) \\ &\Rightarrow \forall i \text{ Available}(d, s_i) \rightarrow T] \end{aligned} \quad (5.10)$$

$$\begin{aligned} DPDM_CheckIn(d') &\Leftrightarrow [d \in db_i \mid d \rightarrow d' \wedge DPDM_CheckOut(d) \\ &\Rightarrow \forall i \text{ Available}(d, s_i) \rightarrow T \wedge Register(d')] \end{aligned} \quad (5.11)$$

5.3.6 Release in DPDM System

The release function restricts further modification to a data object. In Figure 5.11, the sequence diagram shows that the data object cannot be modified anymore once it is finalised and approved by authorised users. The local PDM server executes the *NoUpdate*(d) function to modify the status of the data object in meta-databases at all the remote PDM servers.

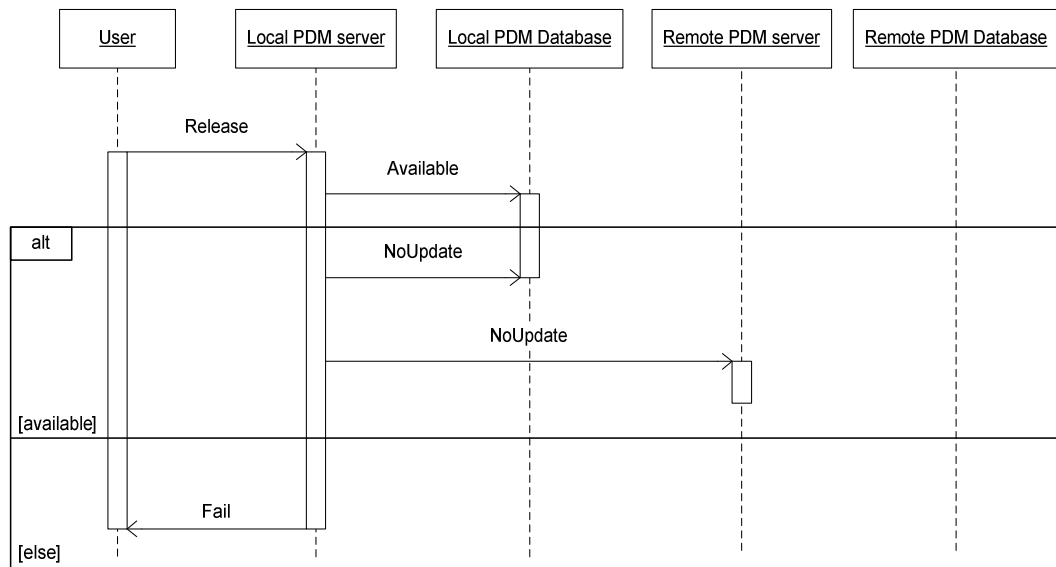


Figure 5.11 Sequence diagram for releasing objects in DPDM system

The formula of release function is required to ascertain that the user is eligible to hold the ownership of the data object d . If d is not occupied, then it can be released. That is, all copies of this object become non-modifiable once they have been released.

$$DPDM_Release(d) \Leftrightarrow [Available(d) \Rightarrow \forall i Modifiable(d, s_i) \rightarrow F] \quad (5.12)$$

5.3.7 Obsolescence in DPDM System

The obsolete function in DPDM system asserts the occupancy of the data object on which the obsolete function is acting. Once the data object is an obsoleted state, all corresponding information of this object cannot be used in any project thereafter. Figure 5.12 is the sequence diagram showing the process involved in the obsolete function.

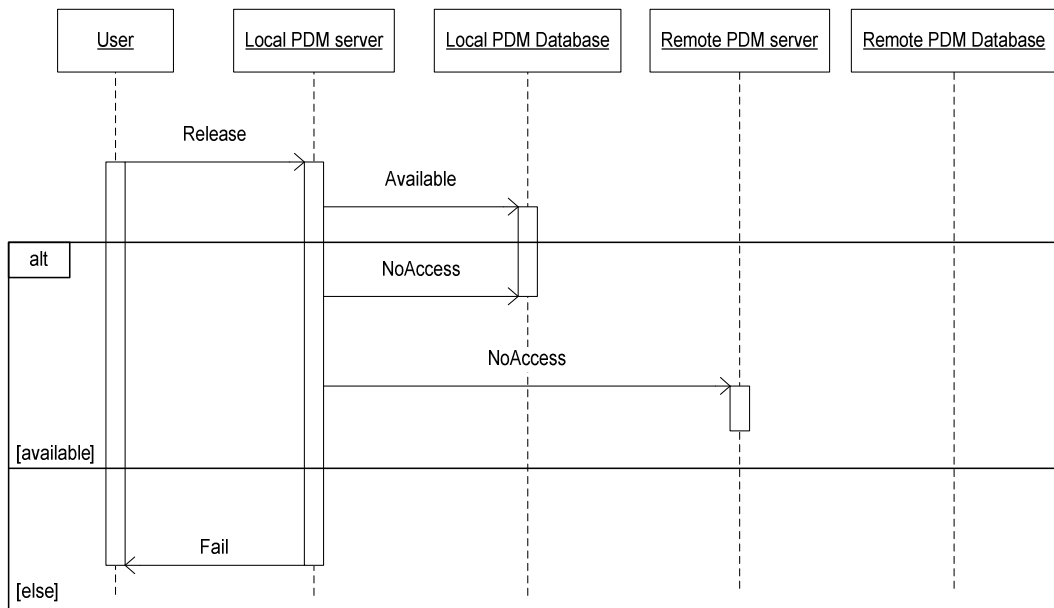


Figure 5.12 Sequence diagram for obsoleting objects in DPDM system

The slight difference of the obsolete function between the centralised and distributed PDM system is that the local PDM server in DPDM system will update the new status of the object in the meta-database in all remote servers, so that the users connecting to the other PDM servers will know that the data object has been obsoleted at the moment they access it.

$$DPDM_Obsolete(d) \Leftrightarrow [Available(d) \Rightarrow \forall i Available(d, s_i) \rightarrow F] \quad (5.13)$$

5.3.8 Deletion in DPDM System

This function erases the chosen data object that must have been obsoleted in the DPDM system. The following sequence diagram illustrates the process of delete function in DPDM system. Determining the selected data object already obsoleted is to ensure that it is not included in any other project and is not being accessed by other users, so that the deletion will not induce data inconsistency to the system.

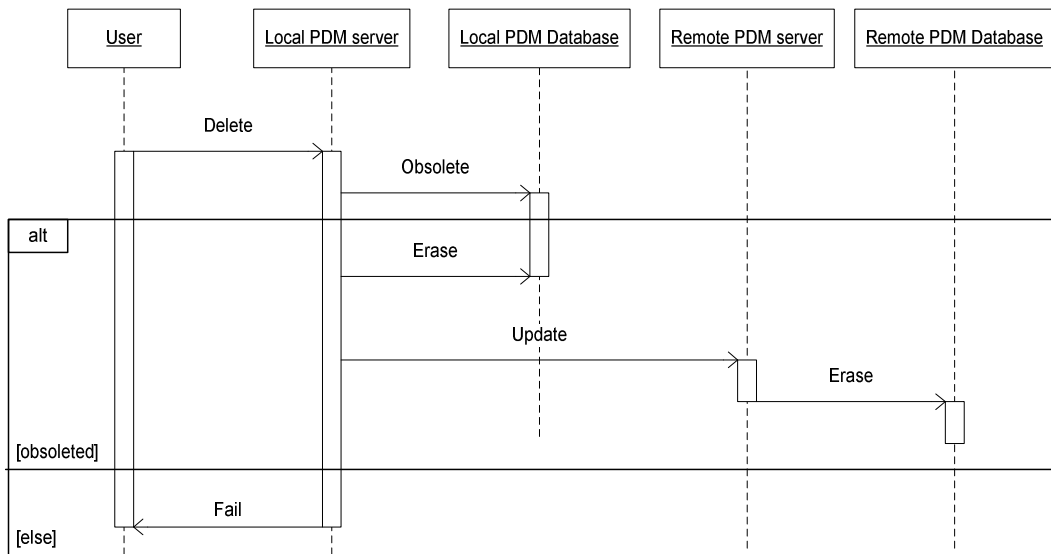


Figure 5.13 Sequence diagram for deleting objects in DPDM system

The following formula is the specification of *Delete* function in DPDM system. It defines that the data object d must be obsoleted. This implies that d is no longer used in any project, then the server will invoke the *Erase* function to delete all the copies of this data object in all PDM databases, and therefore d is no longer an element of any database db .

$$DPDM_Delete(d) \Leftrightarrow [Obsolete(d) \rightarrow \forall i Erase(d, s_i) \Rightarrow d \notin db_i]$$

The user functions of PDM system and DPDM systems are illustrated and defined using the UML sequence diagrams and First Order Logic in this chapter. In the next chapter, common concurrency problems and the specification of a well-known concurrency control in DPDM system is illustrated in the same approach.

Chapter 6

Concurrency Control Specification

6.1 Concurrency Problems in DPDM System

Design and manufacturing workflow can be streamlined by implementing suitable DPDM system that manages all product-related data in an organized manner. One major function of DPDM systems is to maintain data integrity and to provide accurate data when required [Leong, et al. 2003]. The large amount of interactions between the system and the users will give rise to concurrency problems that all data management systems need to overcome.

A transaction is a sequence of actions on some data objects of database. Each action can be categorized into either of the two fundamental database operations: read and write. The read operation returns the content of the data object. The write operation creates a new version of the data object and overwrites the old content or adds to the database if it is newly created. The simple relations among users and a database system when a transaction is executed are depicted using a sequence diagram in Figure 6.1. When the user wants to read the particular data object from the database, he/she selects the data item using the interface provided by the database system. The checkout function is executed to retrieve the data item from the database. The server processes the function by checking the directory to determine the availability of the file. If this data object is available in the database, the server will then transfer it to the user. After referencing the item, the user notifies the server that he/she no longer needs to access it and the connection between the user's computer and the database server will be disconnected. However, if the data item has been modified or is newly created, the user will be prompted to upload it to the database. The server checks the meta-data directory to see if the data object already exists in the database. A new item is written to the database and its meta-data to the directory. Depending on the controlling methods that the system has implemented, the existing data object will

either be overwritten by the newly modified content or a new version is saved to the database along with the old version.

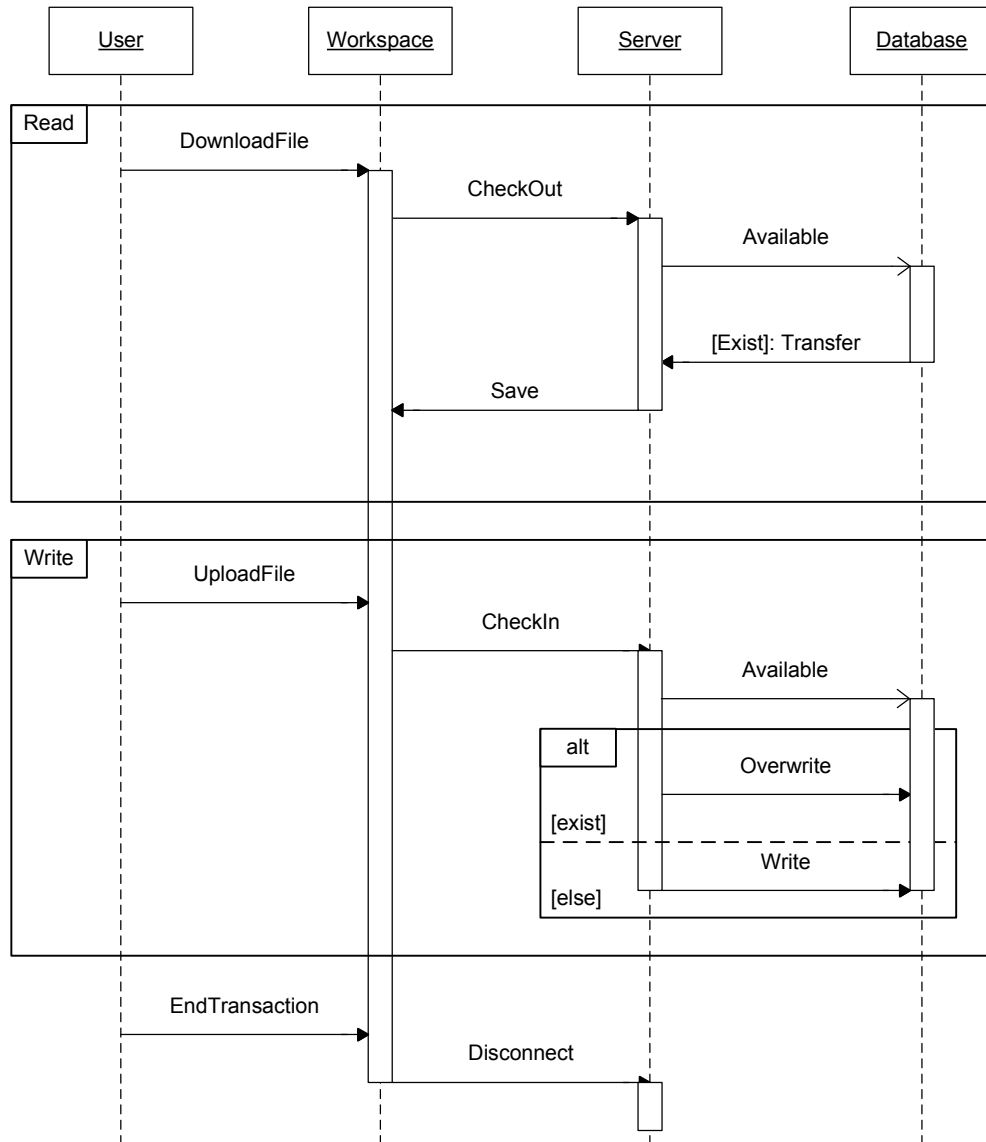


Figure 6.1 Basic database operations: Read and Write

In general, database operations can be executed concurrently within a transaction. Concurrency problems arise when two or more actions are accessing the same data and at least one of which is a write action. [Date 2004] classified the concurrency problems into three ways in which a transaction produces an unexpected result as a consequence of interleaving operations from different

transactions in an inconsistent manner. Two concurrency problems that are related to DPDM systems are illustrated by the following examples in UML sequence diagrams. Sequence diagrams are read from the top to the bottom and left to right.

6.1.1 The Lost Update Problem

Let Opt_i be an operation from the i^{th} transaction of a PDM system, T_i , performing on a file is either a read, write and undo process $Opt_i = \{R_i, W_i, U_i\}$. The lost update problem is presented by means of a sequence diagram in Figure 6.2. The example begins when data object d receives a read operation R_p from transaction T_p at time t_1 , activation lines are rendered on the lifelines of T_p and T_q to indicate that these objects are active. T_q accesses the same file at time t_2 ; write operations W_p from T_p and W_q from T_q update the file at time t_3 and t_4 respectively. The diagram clearly shows that R_q cannot read the update by T_p , and therefore the work done by T_q does not base on the content at time t_3 , instead it worked on the file seen at time t_2 , which are the same as those seen at t_1 . Since W_p is processed after R_q , the update of W_q indeed would not contain the work of W_p , the update by T_p is lost at time t_4 when W_q is committed.

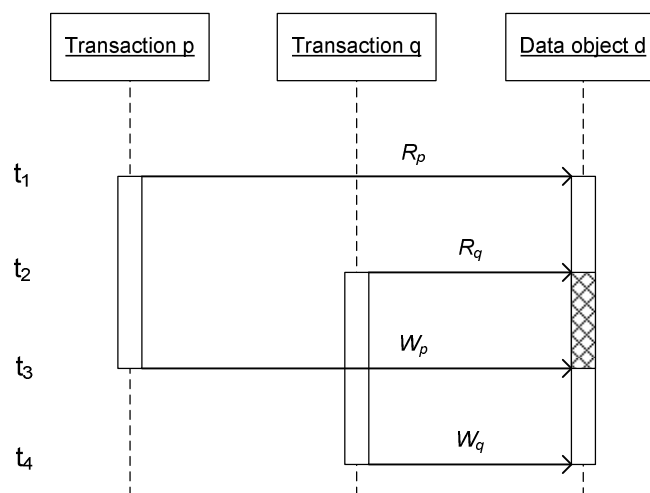


Figure 6.2 The lost update problem

6.1.2 The Dependency Problem

Consider the situation illustrated in Figure 6.3. Transaction T_p updates data object d at time t_1 and transaction T_q reads the update at time t_2 . That update is then undone at time t_3 . As the consequence of U_p from T_p , T_q has retrieved a data object that now no longer exists after time t_3 . As a result, T_q might produce incorrect output, because the value it had is before time t_1 . The problem becomes more complicated if other transaction has updated the file before the undo is processed. In the second example shown in Figure 6.4, T_q reads the file at time t_2 that was updated at time t_1 by T_p and updates it by write operation W_q at time t_3 . W_q become dependent on W_p , however it is discarded with the undo operation U_p at time t_4 in which T_p is supposed to undo its write operations W_p . The dependency problem turns out to be another version of the lost update problem.

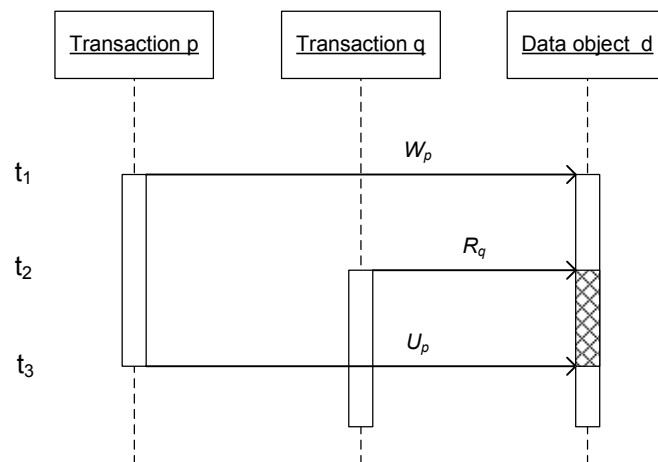


Figure 6.3 Dependency problem: Referencing an undone data object

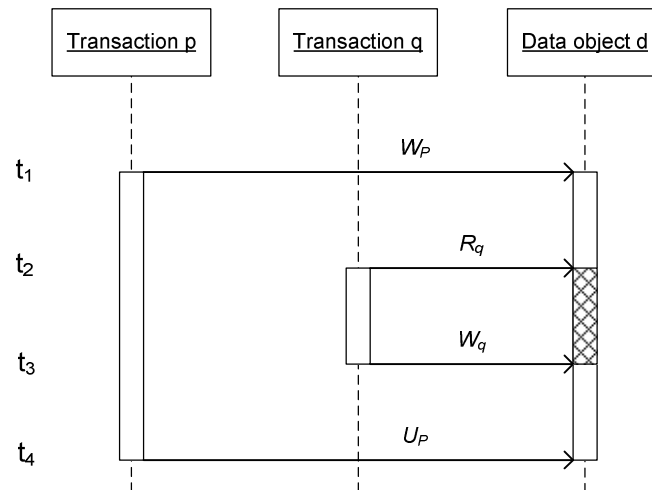


Figure 6.4 Dependency problem: Undoing the work of other transaction

6.3 Basic 2PL Protocol in Sequence Diagram

From studying a number of proposed algorithms, the literature review suggested that they are developed on the basis of two well-known mechanisms for controlling concurrency: two-phase locking (2PL) and timestamp ordering (T/O). The 2PL approach introduced by Eswaran [Eswaran, et al. 1976] is chosen as the basis of the proposed concurrency control model. The reason for choosing 2PL to illustrate the proposed specification approach is that it is a more appropriate concurrency method for DPDM systems. Unlike real-time systems, the activities in DPDM environment often have long lifespan, abortions of transactions incurred from any violation of the time orders in T/O models are intolerable.

Two-phase locking method preserves database consistency by avoiding read and write operations processed from different transactions concurrently on the same data object. A transaction must obtain a read-lock or a write-lock on a data object before starting the read or write operation respectively. Conflicts occur when a transaction requests a lock on a data object that has been locked; the request will not be granted under the following two situations: 1) one is a read lock and the other is a write lock, and 2) both are write locks. Figure 6.5 is a UML sequence

diagram illustrating the 2PL method. Database operations are omitted from the diagram for simplicity. It is also necessary for each transaction to lock each data object before acting on it.

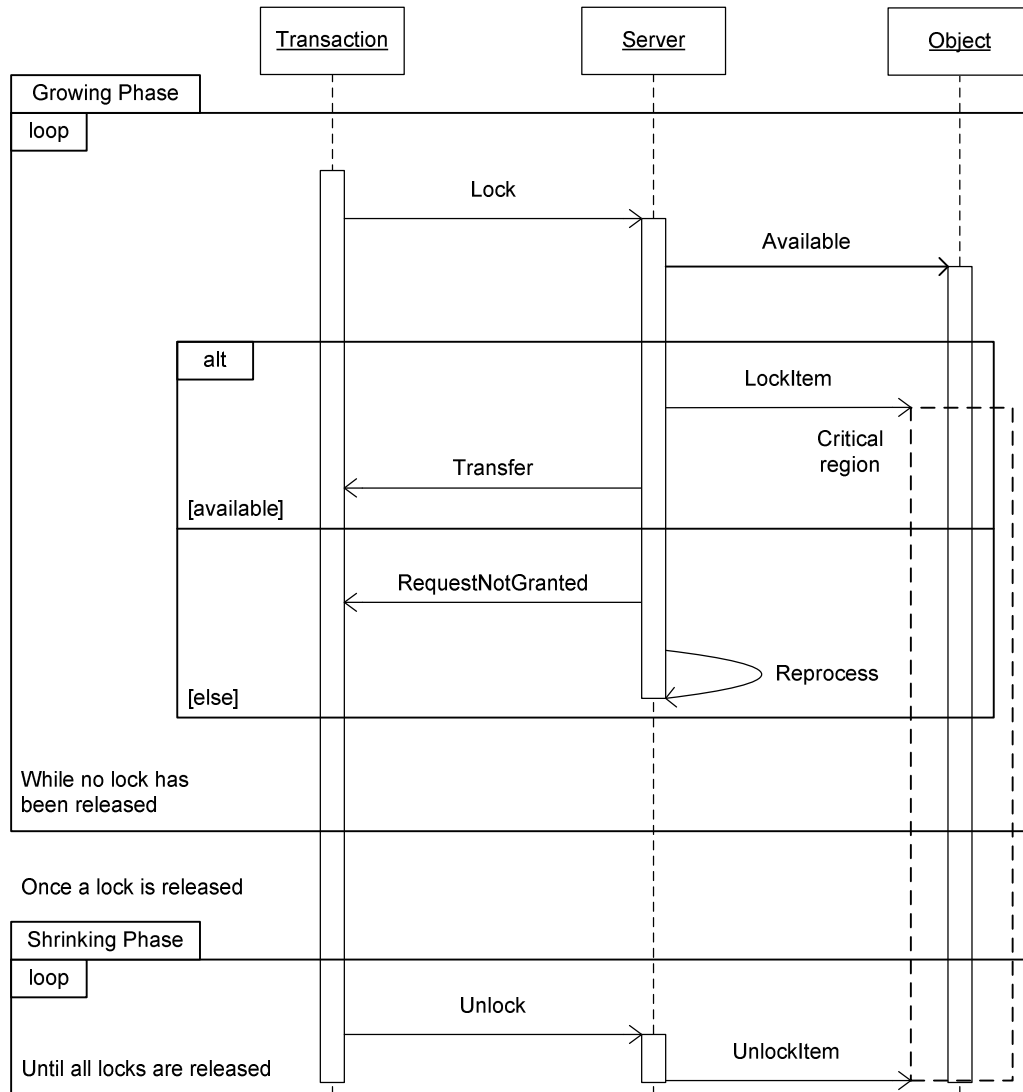


Figure 6.5 Basic two phase locking protocol

In the growing phase, a transaction can request locks on the available free data objects. If the data object requested has already been locked, the request will be reprocessed. 2PL forces transactions to wait for the unavailable locks. If this waiting is uncontrolled, deadlocks will be arisen. The shrinking phase of a transaction begins when one of its locks has been released. During this phase, the transaction cannot obtain any more locks on any data object. Consistency is

established by dividing the locking procedure into two phases. The first phase is growing phase as the transaction is allowed to request locks. By releasing a lock, the transaction enters the shrinking phase. In this phase, the transaction cannot obtain additional locks. Eswaran gave an example where a non-two-phase schedule may lead to inconsistency and suggested that 2PL is a sufficient condition for preserving database consistency. In the case of transactions which are not processing any data object in common, they can be scheduled consistently in any order without violating the consistency, since there is no interaction in between.

6.4 Formal Specification of Concurrency Control

Formal specification has to be unambiguous so that system developers can understand the requirements and develop a system that operates accordingly and able to verify that the specifications do not have any contradiction which would lead to inconsistency. It is widely agreed that a natural language cannot be considered as a good specification language. This is due to the fact that computers are not capable of understanding the meaning of natural language. Meanwhile, classical formal methods, namely mathematical proof, are not widely accepted in the industry since there are too many streams that people can use to specify a system, although classical methods guarantee the correctness of a system without exhaustive tests [Drusinsky 2006].

6.4.1 Temporal Logic

The branch of temporal logic chosen to describe the concurrency model in this research is Propositional Temporal Logic (PTL) introduced by Pnueli [Manna & Pnueli 1992]. Other than the standard propositional connectives \neg (not), \wedge (and), \vee (or), \oplus (exclusive or) and \Rightarrow (implies), the typical temporal operators used to construct the formulas of the 2PL method are as follows:

$\bigcirc p$	Next: In the next moment in time that p will be true.
$\bullet p$	Previous: In the previous moment in time that p was true.
$\square p$	Henceforth: For all future time p is true.
$+ p$	Has been: from the preceding moment in time that p is true (including now)
$\diamond p$	Eventually: At some future time p is true.
$\blacklozenge p$	Once: p holds at some preceding position.
$p U q$	Until: p is always true until the time when q becomes true
$p W q$	Unless: $(p U q) \vee (\square p)$

6.4.2 Integration of Sequence Diagram and Temporal Logic

This section illustrates the syntax and describes the semantics of the integration of temporal logic and sequence diagram. The UML sequence diagrams in previous sections are shown for these purposes. The construct of the temporal logic formulas is based on the order of the function predicates being asserted. A function predicate can be asserted to hold at any state of the lifespan of a data object. Unless function predicates are refuted, they will remain hold from the state at which functions are evoked. Consider the lost update problem shown in Figure 5.2, transaction T_p evokes the read function R_p , $Read(T_p, d)$ is asserted to hold until the transaction evokes the write function W_p that changes the properties of d , the state of R_p is no longer hold because T_p cannot read and write the same data object simultaneously. However, the transaction evokes a new function processing the same data object does not necessarily change the state of the previous function predicate given that earlier function predicates are not conflicting with the newly evoked functions.

6.4.3 Representation of Serializability in Temporal Logic

The notion of locking is the mechanism of 2PL for preserving database consistency in distributed environments. Locking protocol ensures that transactions access data objects serially by blocking later conflicting operations. That is, the operation of the earlier transaction must be completed before the operations of other transactions start if they are incompatible and the transactions are in no way interleaved.

Serializability can be achieved by executing a set of concurrent transactions one at a time in some unspecified serial order. The mutual exclusive formula below is asserted to hold when only one function in the conflicting set will be processed on a data object.

$$\Box \Diamond \oplus F(d) \Leftrightarrow f \in F \quad \Box \Diamond (\neg f_1(d) \wedge \neg f_2(d) \wedge \dots \wedge \neg f_n(d)) \quad (6.1)$$

Formula (6.1) states that it should always be only one function accessing the object, given that $F = \{f_1, f_2, \dots, f_n\}$ is a set of functions that cause inconsistency when processing on the same object concurrently. It is clear that the specification $\Box \Diamond \oplus F(d)$ is true if and only if at most one process is operating on the data object d . A state in which the specification is false is a state in which mutual exclusion is violated.

6.4.4 Specification of Two Phase Locking in Temporal Logic

2PL method ensures that transactions access a data object in a mutually exclusive manner. This protocol results in an additional set of actions called lock and unlock. In the sequence diagram, the data object enters the critical section once locked by a transaction. It represents all the activity that has to be performed without any interference. Non-termination of the critical statement corresponds to one process accessing the data object and never releasing it to the other processes.

$$Critical(d) \Leftrightarrow_+ \exists i Own(i, d) \quad (6.2)$$

A data object d is said to be in its critical section if it is owned by a transaction. A critical region is shown in the sequence diagram when d is locked, this means that only the transaction that owns d is entitled to process it and no other transaction can access the data object.

The logical formulas use two predicates to model conditions of a transaction. The predicate $Lock(T,d)$ is true when transaction T locks the data object d , which also implies that T has been granted the right to access d . Another predicate $Unlock(T,d)$ is true when T releases its lock on d , and it surrenders the right to access d . For defining the actions, the predicate $Own(T,d)$ is introduced to state the ownership of a particular data object. It holds true if T has the ownership of d . The definitions of $lock$ and $unlock$ are as follow:

$$Lock(T,d) \Leftrightarrow [\neg Critical(d) \wedge \neg ShrinkPhase(T) \Rightarrow (Own(T,d) \cup Unlock(T,d))] \quad (6.3)$$

Its premise consists of two predicates: $\neg Critical(d)$ and $\neg ShrinkPhase(T)$, it states that a transaction locks data objects only in growing phase and the data objects requested must not be in critical section. The consequent of $Lock(T,d)$ holds when T is granted the ownership of the data object until the lock is released.

A data object is in critical state when it has been locked by a transaction.

$$\neg ShrinkPhase(T) \Leftrightarrow \blacklozenge(\exists d \ Unlock(T,d) \wedge Own(T,d)) \quad (6.4)$$

Shrinking phase is antonymous to growing phase. That is, a transaction is in shrinking phase then it could not be in growing phase. Therefore either one of two phases is needed to be defined in order to indicate the status of a transaction. The definition states that the transaction is in shrinking phase once it releases all of its locks.

$$Unlock(T,d) \Leftrightarrow [+ Own(T,d) \Rightarrow \square \neg Own(T,d)] \quad (6.5)$$

It is obvious that the transaction must have locked the data object d and have the ownership of d in the pervious instant of time. When T unlocks d , it gives up its ownership of the entity.

$$\text{Transfer}(d, ws) \Leftrightarrow \blacklozenge \text{Lock}(T_k, d) \wedge \text{Issue}(ws, d) \Rightarrow \text{Copy}(d, ws) \quad (6.6)$$

This function ensures that an entity is transferred to the workspace from which the transaction is issued. Data object d is transferred to workspace ws once locked by transaction T_k , given that T_k is issued from ws .

$$\text{Repeat}(\text{Lock}(T_k, d), \text{Unlock}(\mathcal{T}, d)) \Leftrightarrow [\text{Lock}(T_k, d) \cup \forall j \in \mathcal{T} \text{Unlock}(T_j, d)] \quad (6.7)$$

The function repeats the locking action until T_k locks the data object. The requesting transaction will not be able to lock the data object until all other transactions that have locked it release their lock. The action of locking a data object will repeat, given that the transaction cannot lock the data object.

The *Repeat* function executes a set of actions A continuously until the whole set of criteria Γ is met. All kinds of repetitive tasks can be expressed by this formula.

$$\text{Repeat}(A, \Gamma) \Leftrightarrow [A \cup \Gamma] \quad (6.8)$$

The following truth table illustrates the execution of a formula:

$$\text{Repeat}([x \leftarrow x+1] \wedge [i \leftarrow i+1], [i = 4])$$

i	0	1	2	3	4	5
$i \leftarrow i+1$	T	T	T	T	T	T
$i = 4$	F	F	F	F	T	F
x	1	2	3	4	5	6
$x \leftarrow x+1$	T	T	T	T	T	T
$(x \leftarrow x+1) \wedge (i \leftarrow i+1) \vee (i = 4)$	T	T	T	T	T	F

Table 6.1 The truth table of repeat function to perform repetitive addition

A for-loop of adding 1 to the variable x in 5 iterations is expressed using *repeat* function. The pre-conditions of the for-loop are not included in the formula for simplicity, x is five and i is zero as the loop starts. The loop stops when the iterator i equals to 4. Although the assignment functions $x \leftarrow x+1$ and $i \leftarrow i+1$ always hold, the specification of the formula $[x \leftarrow x+1] \wedge [i \leftarrow i+1] \vee [i = 4]$ no longer holds when the iterator will not be equal to 4 once it was reached. Thus, the function *repeat* terminates as it is evaluated to false.

A state of the system is defined by an assignment of truth values to the function predicates. The consistency of the system can be verified by evaluating the sequence of the function predicates. The valid sequence of states of a system is described by the temporal logic specification for the system. The precise conditions represented by these formulas of the specification depend on the details of an implementation. Suppose transaction T has locked the data object d , and there is another transaction trying to access d , a formula is needed to determine that d is not being accessed by any transaction. The state of the function can be either specified as $+Lock(T, d)$ to indicate that d has been locked by T or as $Lock(T, d)$ to indicate that d is locked by T in the very instant of time when checking is preformed. In the proposed approach, representation of specifications in simple form is preferred over complex ones.

Developing system requirements are not an easy task. This chapter described the use of temporal logic in interpreting the UML sequence diagram. In particular, the integrated approach was illustrated through expressing the specifications of the

basic two-phase locking. In this way, system developers can enjoy the simplicity of UML while the correctness properties of models can be formally proved. By having a sound theoretical basis, concurrency control models can be simulated first before implementation. The description of new models can be tested for correctness and optimized in performance. The following chapter describes a concurrency control model that integrates the concept of granularity and versioning. The model is designed on the basis of the specifications of two phase locking protocol and the objective is to improve the concurrent accessibility of a DPDM system.

Chapter 7

Concurrency Control Method for DPDM System

Many variations of concurrency control schemes have been introduced to improve concurrency and system performance in conventional database environment, but an adoption of any of these methods would not be an ideal solution for resolving concurrency problems of a PDM system. Above all, the differences between the natures of DPDM systems and conventional database systems limit the efficacy of these methods.

Locking all the data related to the objects for making changes reduce the degree of concurrency or overall system throughput. One factor that influences PDM concurrency is data complexity. The more number of components of an object has, the data is more likely to be amended by more users. A procedure that can isolate only the affected parts of the data will be helpful to conduct PDM with efficiency. The approach presented in this chapter introduces several types of lock based on the data complexity and the user's actions.

7.1 Formal Description of the Method

The proposed hybrid concurrency control model for DPDM integrates granularity locking and versioning technique. For the former, the model considers each assembly to be divided into different smaller assemblies that are composed of a number of parts. The model also assumes that when a user updates an assembly, a user checks out the lowest level assembly, that is, the direct assembly to which the targeted object belongs rather than checking out a large assembly with many assemblies in the writing process. However, this assumption is not applicable in the case of reading an assembly. For version management, a new concurrency control model that integrates locking granularity with version control is devised. In this model, whenever a user

checks out (updates) a data object, a new version of the data object will be created. No other users are allowed to check out the new version while the current version can still be read by others. After the user has completed the update, he/she then checks in the new version of file back to the vault. Other users can now check out the latest version of the file from the vault.

7.1.1 Locks with Version

Three standard modes of lock [Gary, et al. 1975], Shared (*S*), Exclusive (*X*), and Intent Shared (*IS*), are adopted in this model and two new modes of lock, Version (*V*) and Intent Version (*IV*), are introduced to support versioned concurrency control of a data object during its update process. The server to which the workspace of the user is connected will lock the related data objects using one of those locks according to the operation of the user. The availability of the data objects is determined by the compatibility of the applied lock. Each transaction can hold exactly one lock on each data object. The parts (leaf nodes) can be locked in *S*, *X* or *V* mode; *IV* mode is used in level 0 assembly; *IS* mode is used from project to level 0 assembly. The lock granules precedence graph in Figure 7.1 shows how the locks are applied to data objects in different levels of complexity. The vault stores a number of projects and each is composed of various assemblies. Data objects in these levels hold an *IS* lock when they are accessed. The assembly to which the simple parts directly belong is at level 0, where *IV* lock is applied to the assemblies at the lowest complexity level (level 0) and *S*, *V*, and *X* locks are applied to parts that belong to a data object that is accessed by a user.

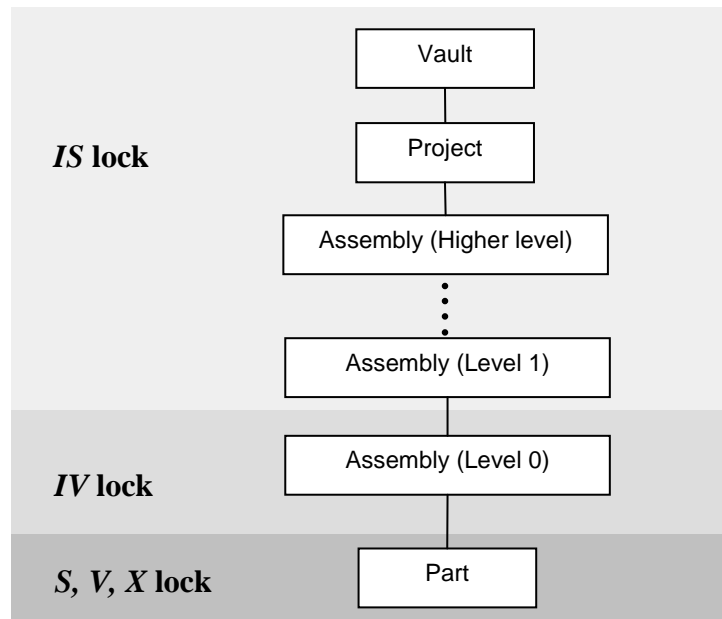


Figure 7.1 Lock granules hierarchy

The principles of locks with version are described as follows.

- *S* lock on a leaf node permits other users to read the same node concurrently but prevents any updating of the node
- *V* lock on a leaf node implies a new version of the part is being created while the current version is still readable by other users.
- *X* lock on a leaf node implies the node is the new version being created by a user at the same time excludes any other user from accessing the node.
- *IS* lock of a data object specifying its descendant parts will be explicitly locked in *S* lock.
- *IV* lock of a level 0 assembly implies that explicit locking is being done on some parts in *V* lock.

For instance, to read the data object d , transaction T_k first locks the direct assembly of d in *IS* lock, and then locks d in *S* lock. To update d , T_k first locks

the direct (level 0) assembly of N in IV lock, and then it locks d in V lock. Locks should be released in leaf-to-root order before the end of a transaction or in any order at the end of the transaction. Table 7.1 is the lock compatibility matrix. It shows that the concurrency power of S is larger than V as two locks are compatible with S while one lock is compatible with V in the request mode. The compatibility of V lock is weaker than S lock, since it is compatible with other S locks only but not with any lock of other types. Lastly, at the part level, X lock is the most restrictive lock as it is not compatible with any lock at all.

		Request Mode				
		S	X	V	IS	IV
Current Mode	S	Yes	No	Yes	N/A*	N/A
	X	No	No	No	N/A	N/A
	V	Yes	No	No	N/A	N/A
	IS	N/A	N/A	N/A	Yes	Yes
	IV	N/A	N/A	N/A	Yes	No

*N/A – not applicable since S, V, X locks are used in parts while IS and IV locks are used in assemblies

Table 7.1 Compatibility matrix for granularity locking

Suppose there are three transactions T_1 , T_2 , and T_3 in the system, T_1 and T_2 apply a S lock on the data object d , which is the version 1 of k . T_1 is going to modify d , so it promotes the lock on d to a V lock. Since d_1 has been locked with a V lock already, the request of a V lock on d from T_2 is blocked. Once T_1 modifies d , a copy of this data object is created, that is d' in this example. An X lock is automatically applied on d' for T_1 . T_3 accesses d , because the object is still V locked by T_1 , so the system only grants it a S lock. T_3 also tries to access d' , however, T_1 has not released the X lock on the object, so the request of T_3 is blocked.

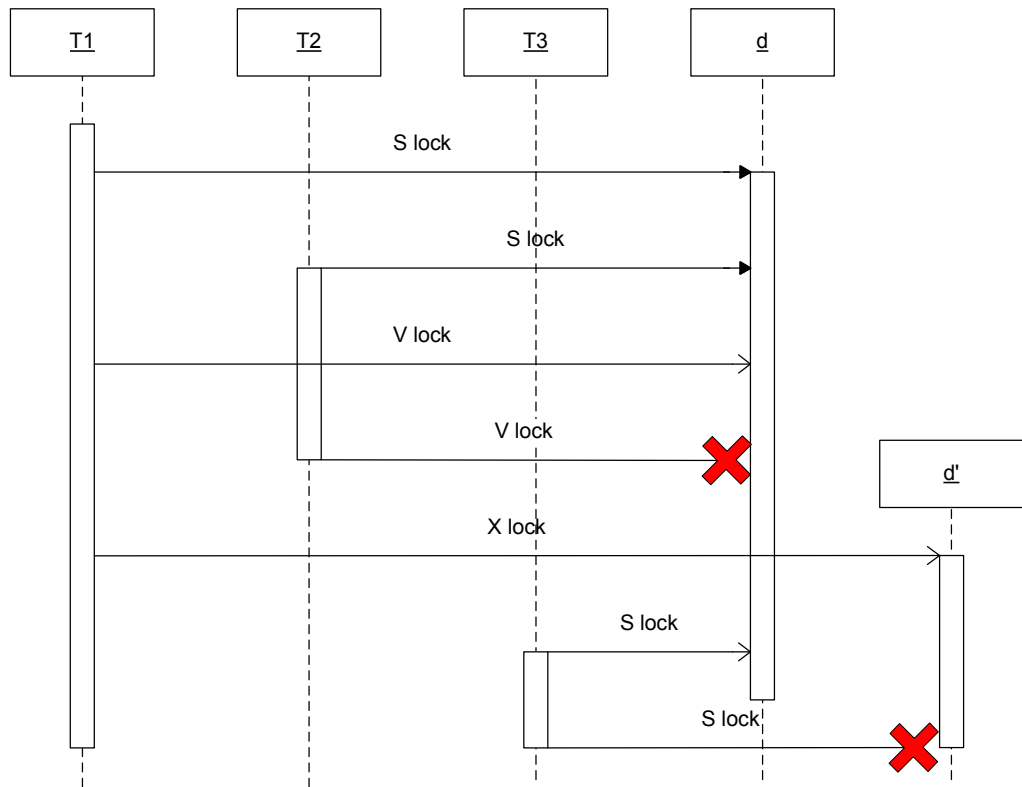


Figure 7.2 Illustration of lock compatibility

For the same reason, the concurrency power of *IS* lock is higher than *IV* lock. The type of lock applied on a data object is determined by the concurrency power in some cases. For example, when a data object already held a lock during transaction T_1 , and a transaction T_2 request is permitted, another lock is going to apply onto the same data object. However, as only one lock can be held by a data object in this model, the concurrency power is used to determine which lock dominates the sharing ability of the data object. The larger the concurrency power, the higher the sharing ability and the higher the potential risk of data lost. In order to maintain system security and to ensure that the transaction mechanism does not conflict with the compatibility matrix in the system, the type of lock chosen may be the one with the less sharing ability.

7.1.2 Notations and Types of Functions

As defined in Section 3.2.2, $\#sd$ has been denoted as the total number of data objects including all assemblies and parts in the DPDM database (vault), assemblies data objects are denoted as $AD = \{ad_1, ad_2, \dots, ad_m\}$ and part data objects are denoted as $PD = \{pd_1, pd_2, \dots, pd_n\}$ where m and n are the index number of assembly and part data objects in the PDM system respectively. Thus,

$$db = \{PD, AD\} = \{(ad_1, ad_2, \dots, ad_m), (pd_1, pd_2, \dots, pd_n)\} \quad (7.1)$$

The following actions are those of the PDM system that can perform on the data objects:

$S(pd)$	Lock part data pd in Shared mode.
$rS(pd)$	Release the Shared lock of part data pd .
$X(pd)$	Lock part data pd in Exclusive mode.
$rX(pd)$	Release the Exclusive lock of part data pd .
$V(pd)$	Lock part data pd in Versioned mode.
$rV(pd)$	Release the Versioned lock of part data pd .
$IS(ad)$	Lock an assembly data ad in Intent Shared mode.
$rIS(ad)$	Release the Intent Shared lock of assembly data ad .
$IV(ad)$	Lock an assembly data ad in Intent Version mode.
$rIV(ad)$	Release the Intent Version lock of assembly data ad .

When a transaction is invoked, one or more of the following functions will be executed.

$Lock(d, L)$	holds if data object d can be locked in lock mode L .
$M(d)$	holds if data object d has been modified.
$P(d)$	returns a set of parts data object that consist of object d .
$A(d)$	returns a set of assembly data object that includes data object d .

A new temporal logic operator called Consequent is introduced for defining the specifications in a more succinct form.

$$C(p, q, r) \Leftrightarrow (\bullet p \rightarrow (p \wedge q) \text{ W } r) \quad (7.2)$$

The above definition says that once p holds, then p and q are always true unless r becomes true.

7.2 Implementation

This section presents the implementation of the model to a DPDM system and how the functions regulate the operations of the system in order to safeguard the integrity of the data. A DPDM system is a platform for making the proper product data available to the right people at the right time. When building database applications, it is not sufficient to install only a database. There must be specific tools that enable a speed-up of data flow and activities. A data controller is built into the DPDM system for managing the access to the data in the system. In a situation when a person issues a transaction to access a file in the system, the data controller will trigger the meta-data processor to determine the files that will be affected by the transaction. The data controller will carry out appropriate actions to the file affected based on the query result. In case a new transaction conflicts with other executing transactions, the new transaction will be stored in the transaction base to wait for the file(s) it requests. The pending transaction will be assessed again when the conflicting transactions have been completed.

7.2.1 Check-out and Release Processes

Checking out or releasing a part data object pd implies that the user wants to update the property of a data object from the vault. As mentioned in the previous section, this model assumes each user to check-out/release the lowest level assembly rather than checking out a large assembly when updating a data object. The model applies the following rule to control the check-out and update operations of the system.

$$\begin{aligned}
 ad \in A(d^r) \quad & C(C(\text{Lock}(ad, IV), IV(ad), rIV(ad)), V(pd^r), rV(pd^r)) \wedge M(pd^r) \\
 & \rightarrow X(pd^{r+1})
 \end{aligned}
 \tag{7.3}$$

The user first locks the direct assembly of that data object in IV mode, and then locks it in V mode. After the version r of the part object pd^r is checked out, a new version of the pd^{r+1} will be created. Concurrently, the IV lock on pd^r is released and the current version (pd^r) is readable by other users. Figures 7.3 and 7.4 illustrate the Check-out/Release process on pd^r with one and more than one direct assembly respectively.

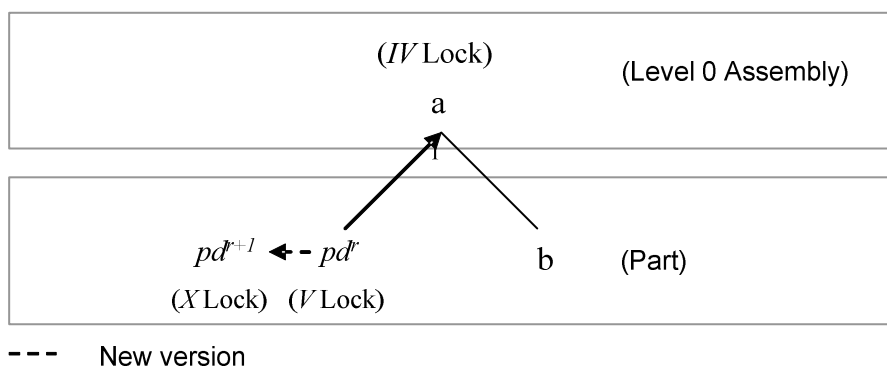


Figure 7.3 Check-out/Release process on part data object pd^r with one direct assembly

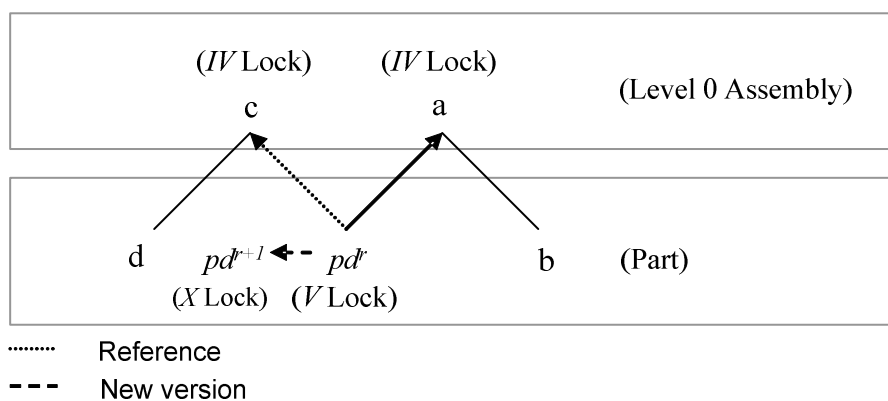


Figure 7.4 Check-out/Release process on part data object pd^r with more than one direct assembly

After the modification to the data object is completed, the new version of the data object is then checked back into the vault and all the locks will be released in leaf-to-root order.

$$ad \in A(pd^r) \quad \bullet(\bullet rX(pd^{r+1}) \rightarrow rV(pd^{r+1})) \rightarrow rIV(ad) \quad (7.4)$$

Simultaneously, the server will notify those users u who have checked out an assembly that contains the modified part a new version is now available. The following is the logical interpretation of the action after modification is done to pd^r .

$$u : \blacklozenge Check_out(ad) \mid ad \in A(pd^r) \quad \blacklozenge rX(pd^{r+1}) \rightarrow Notify(u) \quad (7.5)$$

7.2.2 View Process

To allow a user to view the part data object pd^r , the system first locks its direct assembly in IS mode, and then locks pd^r in S mode. The following definition of transition of locks is illustrated by the locking order of the view process in Figure 7.5.

$$ad \in A(pd^r) \quad C(Lock(ad, IS), IS(ad), rIS(ad)) \rightarrow S(pd^r) \quad (7.6)$$

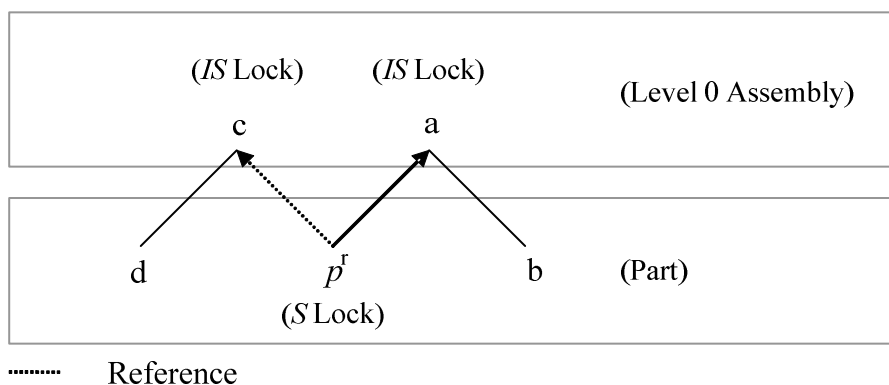


Figure 7.5 Viewing a part with more than one direct assembly

After viewing the part, the system releases the locks in leaf-to-root order. Notice that the following formula uses $_$ symbol (that is, underscore) to indicate the anonymous variable. The third variable of the Consequent operator is an action that will refute the state of the two predicates. Unless either any assembly in set A or the part pd^r is locked by any type of locks, the affected data objects remain unlock from the state at which functions were evoked.

$$ad \in A(pd^r) \quad C(rS(pd^r), rIS(ad), L(_, _)) \quad (7.7)$$

Similarly, to view assembly data object ad^r , the system locks both the direct ancestor and the assembly primarily in IS mode, and then locks its descendants in S mode:

$$ad \in A(ad^r) \mid pd \in P(ad^r) \quad C(C(Lock(ad, IS), IS(ad), rIS(ad)), IS(ad^r), rIS(ad^r)) \\ \wedge C(Lock(pd, S), S(pd), rS(pd)) \quad (7.8)$$

All locks applied on assembly data object ad^r are released also in leaf-to-root order after viewing:

$$ad \in A(ad^r) \mid pd \in P(ad^r) \quad C(Lock(ad, IS), IS(ad)) \rightarrow S(ad^r) \quad (7.9)$$

7.2.3 Obsolete Process

To move a part data object pd to the obsolete vault, the part should not be locked in any mode. The status of the part should conform to the following rule.

$$\neg Lock((S \vee X \vee V), pd) \quad (7.10)$$

Similarly, to obsolete assembly data object ad from the vault, the PDM system must ensure that the assembly and its parts are not locked in any mode. In other words, they are not being used by any users. The following rule checks the condition for all data objects involved.

$$\neg Lock((IS \wedge IV), ad) \quad (7.11)$$

7.2.4 Function of Redlining

Redlining is the visual annotation of CAD files to facilitate the communication between individual PDM users, for example, it reminds the edited places of updated version. Redlining is not a necessity to each CAD file. The model treats this function as an extra component towards the CAD files. The notation of redlining $d^r.RL^i$ denotes the version i of redlining of the version r of the data object d is visible to users. By default, redlining of the file is turned on and set “visible” with version i to remind the PDM user to edit according to the amendment remarks annotating the CAD file. Being a supplementary note to a CAD file, the version of redlining may be different from the CAD file. When undergoing check-out or release process, the version of the CAD file is incremented while the version of redlining remains unchanged if the user does not make any amendment on it. If the version of redlining remains unchanged, the redlining function can be turned off until a new version of redlining is created. New version of redlining will be created by the user explicitly by the “save” action. To include this function into the lifecycle of the PDM files, the

notation for the redlining being turned off will be replaced by d^r and in which the redlining being turned on will be replaced by $d^r.RL^i$. Three possibilities regarding the versioning of redlining are illustrated in the Figure 7.6. The example starts with a data object with redlining $d^1.RL^1$.

1. The user modified only the supplementary note and then the system updates the version of redlining of this data object from $d^1.RL^1$ to $d^1.RL^2$ without changing the versioning of the data object.
2. The user modified the data object without making any changes to the redlining, thus the system updates the version of the object $d^1.RL^1$ to $d^2.RL^1$ but not the redlining.
3. The user modified both the data object and its redlining; the system updates the version of d and the redlining to $d^2.RL^2$.

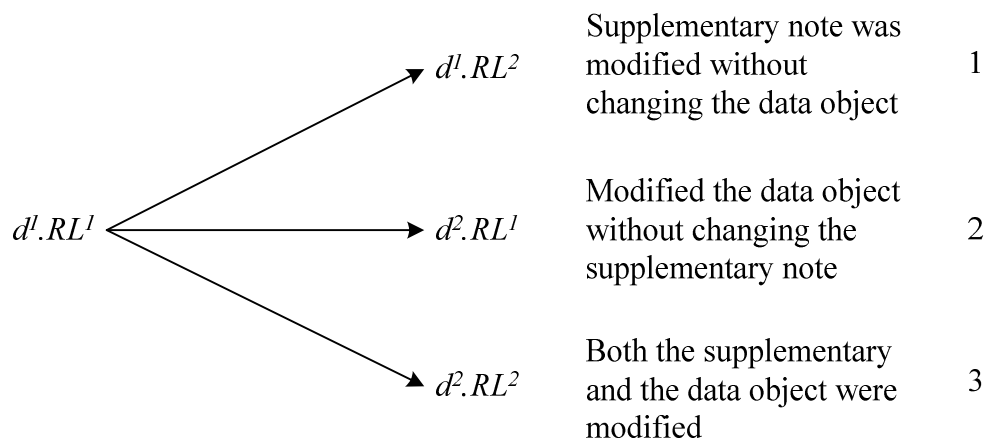


Figure 7.6 Versioning of redlining

7.3 Case Study

The proposed concurrency control model has four classes namely, Object, Project, Part, and Assembly. The attributes of class Object include file name, version, description, redlining, $r_version$, and other information of the

document as shown in Figure 7.7. Attribute redlining stores the visibility of redlining of the data object, attribute *r_version* stores the version of redlining and attribute *version* stores the version of the data object. Class *Part* and *Assembly* are children of *Object*; they inherit the attributes of *Object* and add in some more attributes for themselves. *S*, *X*, *V*, *IS*, and *IV* store the accessibility of *S*, *X*, *V*, *IS*, and *IV* locks. The *part* and *assembly* arrays store the descendants of the current assembly. Class *Project* contains two attributes in array type which store the data objects of the current project.

```
Class Object
{
private:
    // Object information
    string Name[100];
    ...
    string Owner[10];

    int redlining=1; // true=1, false=0
    int r_version = 1;
    int version = 1;
}
```

Figure 7.7 The object class

The implementation of the proposed model will be illustrated via a case study to manage the product data for an ink jet printer production [SolidWorks 2005], which consists of CAD files of parts and assemblies. Figure 7.8 shows a section of the product structure of an ink jet printer. In the example, the ink jet printer consists of three assemblies: ink cartridge assembly, ink jet top assembly, and electronic assembly. Each of them consists of a number of assemblies and parts. In the case study, the focus will be on the ink cartridge assembly.

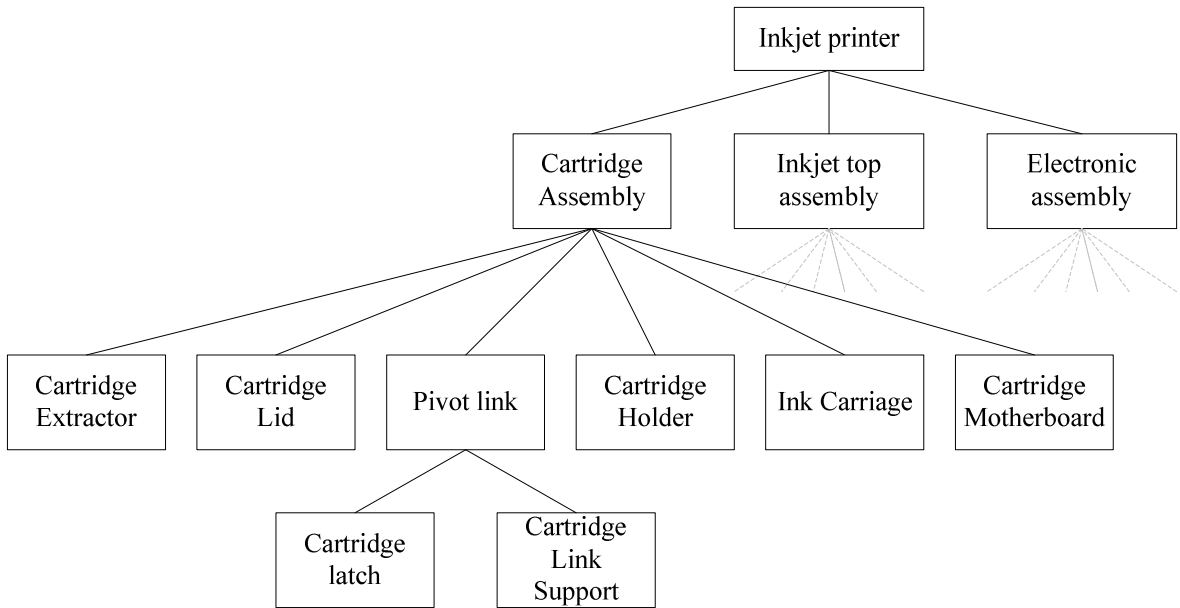


Figure 7.8 Product structure of the ink jet printer

For simplicity, all files of the ink jet printer are assumed to be newly created with version 1. The ink jet printer belongs to the project level, while the ink cartridge and the pivot link belongs to the assembly (level 1) and the assembly (level 0) respectively, and are shown in Figure. 7.9 (a) and (b).

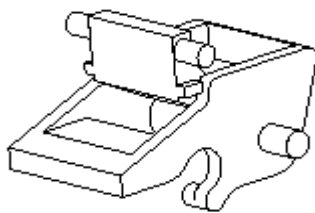


Figure 7.9 (a) The assembly of pivot link

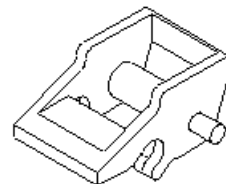


Figure 7.9 (b) The part of ink cartridge latch

Figure 7.10 shows the variety of locks being applied to the data object in accordance with their complexity level when the design of the ink cartridge latch is going to be modified. The process is started by locking the file of pivot link in *IV* mode and the ink cartridge latch in *V* mode. The ink cartridge and the ink jet printer are locked in *IS* mode because *IV* mode can only be used in the

level 0 assembly. After that, the ink cartridge latch with version 1 is checked out from the vault to make modification. A file of the ink cartridge latch with version 2 will be created and will be locked in *X* mode. All modifications of the design are made in this file. Ink jet printer, ink cartridge, pivot link and ink cartridge latch with version 1 are still readable by other user except ink cartridge latch with version 2.

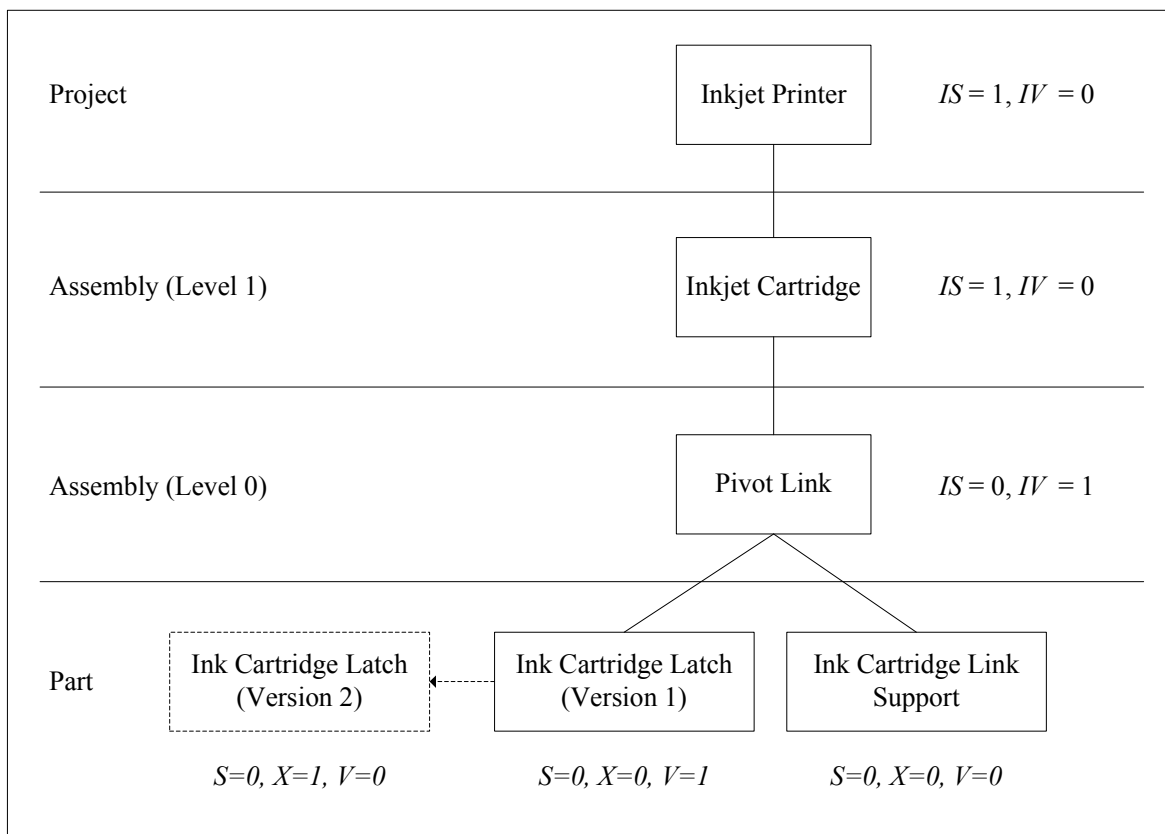


Figure 7.10 The design of the ink cartridge latch is being modified

To illustrate the concurrency ability of the *IV* locking mode, suppose a user wants to modify file B, the design of the link cartridge lid, while file A, the design of the pivot link is under modification. Since the file of ink cartridge, the antecedent of file A, is locked in *IS* mode, its direct descendant other than file A can be modified by others. Although it has already been locked in *IS* mode, the locking mode on this file is now converted to *IV* mode because its lock locking properties dominates. The process of locking mode conversion on a file when there are more than one direct descendants being modified

concurrently is shown in Figure 7.11. The file of link cartridge lid with version 1 is locked in *V* mode and a file of version 2 is created which is locked in *X* mode for modification. After the modification of the ink cartridge lid, file with version 2 is checked back into the vault and all the locks are released in leaf-to-root order. The notifications to the direct assembly of the ink cartridge lid are triggered (ink cartridge and ink jet printer) to inform the users that a new version of the part is now available. In the meantime, as the file of ink cartridge is no longer affected by the *IV* lock, it will change back to *IS* lock. Finally, the locks on the data objects involved will be released when all the above modifications are completed.

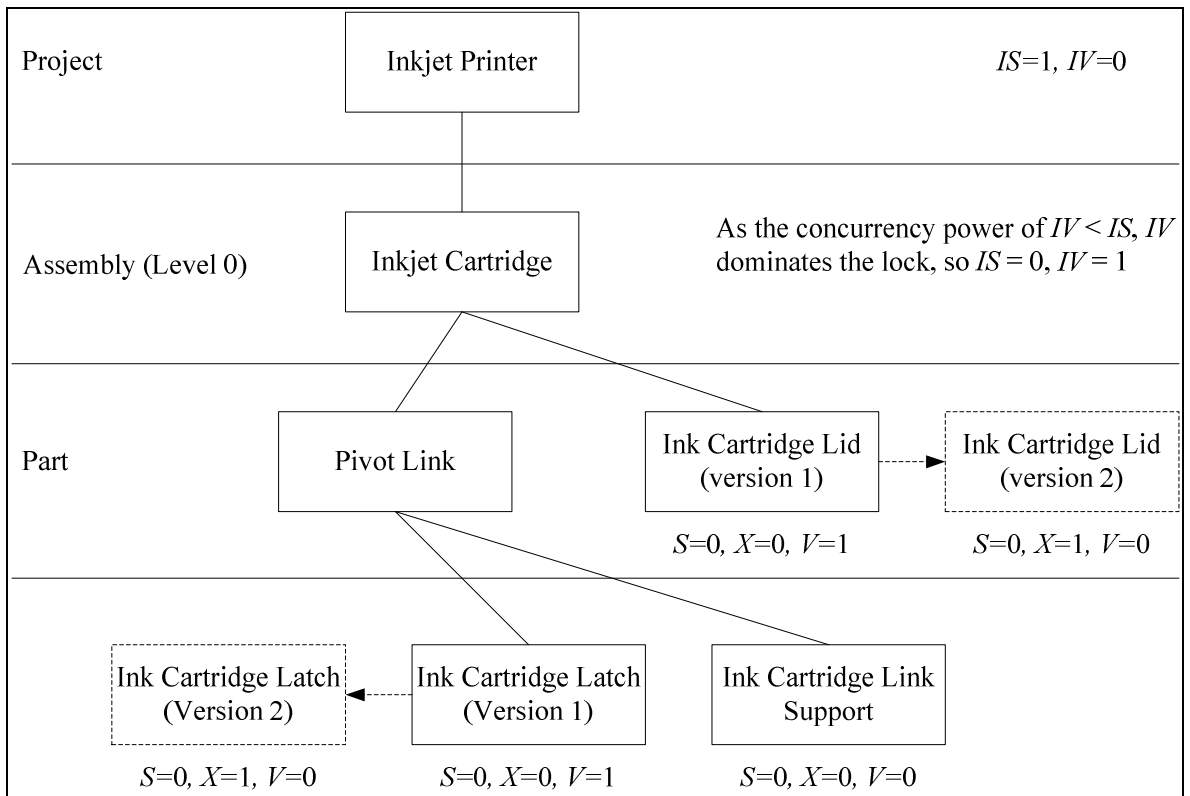


Figure 7.11 The design of ink cartridge lid is being modified

In product developments, product data are mainly composed of the information of assemblies and parts, which are often managed in distributed computing environments. DPDM systems are often utilized for managing the data access. When modifying the design of a product, a number of independent tasks may

be performed to different components of products. However, locking all corresponding files and their parts with one single lock limits the concurrency of the DPDM system. The efficiency of a product development will thus be slowed down.

A new concurrency control model is presented in this chapter that improves the concurrency ability of DPDM systems by adjusting the accessibility of data objects in accordance with the action to be performed by the users and the product architecture of the physical entity. The model allows more simultaneous access to the product data by switching the type of locks being applied. However, there is a trade-off on the number of concurrent accesses and lock conflicts. The finer the granularity, the greater the overhead on lock testing will be.

A methodology which enables the incorporation of a scheduling technique and integrated locking method to improve the performance of DPDM systems will be presented in the next chapter. The efficiency can be further enhanced by considering factors that affect the accessibility of product data when granting access permissions to transactions when constructing the transaction schedule.

Chapter 8

DPDM Deadlock Avoidance

8.1 Transaction Scheduling Problem in DPDM System

Concurrent access to a database system is a way to increase the flow of information. Many concurrency control methods have been proposed over the last few decades, but an adoption of any of these methods would not be an ideal solution for resolving concurrency problems of a DPDM system. Above all, the differences between the natures of DPDM systems and conventional database systems limit the efficacy of these methods. It is because these methods were purposefully designed for managing conventional database systems. Furthermore, these methods are absent from taking the account of durations and deadlines of transactions and the precedence of execution of transactions. However, engineering applications often require consistent and long-term detainment of large volume of data and meeting project deadline is crucial to the success of the business. It is therefore important to develop a concurrency control mechanism that can incorporate with the ability of scheduling in order to increase the concurrent access of the PDM systems while data integrity is maintained.

While improving concurrency of a PDM system is a critical factor in facilitating a fast information flow, a tool for producing the best schedule without sacrificing the data consistency is developed. In the proposed transaction scheduling model, the basic unit of product data granularity considered is the data object. A transaction accesses a set of data object. If the set of required data objects are all ready for access, the transaction holds lock of these files and will be processed, otherwise the transaction has to wait until all the files are lock-free. The locking technique employed in the proposed model has been described in Chapter 6. Read lock restricts the data to be read only by the transaction that applies this lock and exclusive lock allows

transactions to both read and write to the data locked. Exclusive locks cannot be applied to the data that have already been locked by other transactions; conversely, no other transactions can apply any lock to the exclusively locked data object. This will certainly guarantee that no concurrent transactions will be able to update these data before the locks are released.

8.1.1 Problem of Deadlocks

Maintaining the integrity of a database is of crucial importance in a shared environment. This goal can be achieved by producing a serializable schedule of transaction executions [Date 2004] and graph theory is employed to determine the serializability of schedules [Eich 1988]. Locking is one of the well-known concurrency control technique and more likely to be encountered in practice. A transaction can obtain a lock on a data by issuing a request to the system and perform appropriate actions depending on the lock type. The basic idea is that when a transaction T needs an assurance that some data objects it is interested in will not be altered, thus T acquires a lock on these data objects. The effect of acquiring the lock is to prevent other transactions from changing the data objects in question. However, locking has the risk of deadlock as the transactions may wait for unavailable locks [Philip & Nathan 1981]. Although locking guarantees serializable schedules, it is not necessarily deadlock free. A deadlock occurs when a set of two or more transactions are requesting data locked by others in the set. Thus, these transactions will wait to be executed forever if none of them is cancelled. Thus, deadlocks must not exist to ensure that every transaction will eventually be executed. The main approaches for resolving deadlock are deadlock detection and deadlock avoidance. The conflict resolution employed in the proposed model is called transaction scheduling. This approach involves scheduling transactions for execution in a way that two transactions will not be processed concurrently if a deadlock will occur.

8.1.2 Definition of Transaction

Let $\tau = \{T_1, \dots, T_n\}$ be a set of transactions to be executed in a DPDM system. Each transaction $T = \{d_1, d_2, \dots, d_n\}$ requires a set of atomic data d to be executed. Atomic data are data that ought not to be granularised any further. Transactions can be divided into two categories: mature and pending transactions. The former is a transaction that every data object required for the process is available upon request and can be executed immediately. Conversely, the latter are transactions that have to wait as one or more required data objects are locked by other transactions.

Each transaction has a latest start time lt , a deadline dt , and an estimated processing time pt . The latest start time is the time that the transaction should be processed for not missing its deadline. The latest start time of a transaction is $lt = dt - pt$. The deadline is the time at which the transaction should be completed. These parameters are known to the system when the transaction arises. However, the action of transactions upon the data objects may not be determined. The type of locks on which the transactions applied is set to be read locks for the sake of concurrency unless users specify that they are likely to change the content of the files during the transactions. Methods for improving the concurrency of a system will be discussed in later sections.

8.1.3 Deadlock Avoidance

The methods adopted by the transaction scheduling model to prevent the occurrences of deadlock and minimise the tardiness of transactions are described in this section. A transaction scheduling problem for DPDM system can be depicted as a graph. The source node S is the DPDM system which contains all the data, where the transactions and the precedence of the execution are represented by vertices and directed arcs respectively. The direction of the edges indicates the dependency between the vertices. The vertex at the head of the arc requires some data files from the vertex at the tail.

The sequence of execution of transactions is represented by passing data objects from one node to another until all transaction nodes have been passed. Suppose a set of transactions, $\tau = \{T_1, T_2, T_3\}$ is to be executed, where $T_1 \cap T_2 = \emptyset$ and there are some data objects which are accessed by both T_1 and T_3 , that is, $\exists d \ d \in (T_1 \cap T_3)$, where the action of one transaction conflicts with the action of the other. The directed graph corresponding to the situation is depicted in Figure 8.1. In this example, there are two possible sequences of execution. Suppose T_1 is going to modify the content of the data object d and T_3 requires reference to d . Since writing and reading functions are conflicting functions if performed concurrently, T_1 and T_3 cannot be executed concurrently or the serializability will be violated and the database may become inconsistent. Therefore, these three transactions must be executed in either one of the following two sequences to preserve consistency. The first alternative is to execute both T_1 and T_2 , and then T_3 or to execute both T_2 and T_3 , and then T_1 . The first sequence incurs a total cost $(c_{s,T_1} + c_{T_1,s} + c_{s,T_3} + c_{T_3,s} + c_{s,T_2} + c_{T_2,s})$ and the second sequence incurs a total cost $(c_{s,T_3} + c_{T_3,s} + c_{s,T_1} + c_{T_1,s} + c_{s,T_2} + c_{T_2,s})$. The best sequence of execution is determined by finding a route that goes through all the nodes once with the minimum total cost, where the cost from node i to node j is $c_{i,j}$.

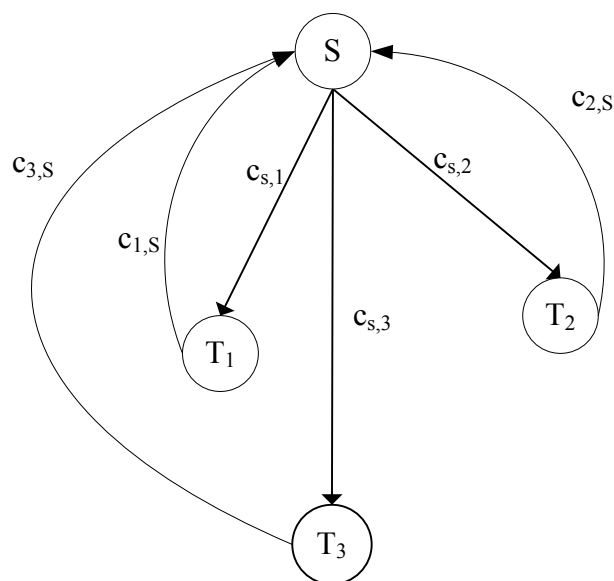


Figure 8.1 Execution precedence graph

An optimal solution to the problem can be computed by finding a cycle that goes through all the vertices with the least total length. Problems of such cycle for networks are known as Travelling Salesperson Problem (TSP) and no efficient algorithms have been developed to solve it. However, determining the complete cycle is not necessary in transaction scheduling for DPDM systems. This is because new transactions may emerge any time, the priorities of new transactions may be higher than those existing transactions. The solution is no longer valid and a new schedule has to be computed and the effort of determining the schedule of the waiting transactions would be wasted.

8.1.4 Objective of the Model

The main objective of transaction scheduling is to minimise the number of missing deadlines. Some transactions may unavoidably miss their deadlines. Subsequent actions have to be taken to handle these transactions. For the nature of transaction in DPDM systems, a transaction that misses its deadline will not become worthless. All transactions must be completed even though they are tardy. Tardiness of transactions is considered to be the secondary objective in which the proposed model minimises. The priority of tardy transactions becomes an issue for the system; there are at least two alternatives available. The first option is to consider that the tardy transactions should receive higher priority to other transactions, as they should be completed as soon as possible. Secondly, these transactions can be processed in later time, since they already missed the deadline and their urgency diminished.

In order to improve the concurrency and efficiency of a DPDM system, the following objectives of the transaction scheduling model are posed to ensure reasonable and correct decisions are made:

- Minimise the total number of deadlines missed
- Minimise the tardiness of late transactions

Firstly, the schedule should include as many transactions as possible, to ensure the transactions are processed before deadline. Secondly, it must consider the concurrency of a DPDM system. These two goals often conflict with each other, as trying to meet one goal will worsen the quality of the other. The objective of the model is to create the best quality transaction schedule, where the quality of a schedule is determined by the level of divergence from the target of each of the two goals. The following features were deemed to be factors that affect the quality of a schedule:

1. Complexity of product development

The time required at an early stage of developing a new product is inevitably longer than to fine tune the product at a later stage. Modifications are made frequently to the design, thus it is likely that files are locked in exclusive mode and concurrent access cannot be exercised.

2. Stage of product life cycle

The rate of retrieval and modification of files depends on the stage of the product life cycle. CAD files and specifications are retrieved and updated frequently in the beginning of the cycle. However, these files are often retrieved for referencing in the production stage and updates are rarely made. Conversely, production line capacity reports of production plant are less relevant to a product in designing phase, but the reports would be accessed frequently when the product comes into production.

3. Revision and modification

It is possible to modify existing parts for designing a new product. Then a newer version of the parts is created and transactions that access the earlier version of the files would not be affected. In contrast, when the part needs

to be revised, all the associated files must be exclusively locked and no transaction is allowed to access these files. Transactions of revision would be assigned a cost, which is determined based on the number of transactions that are waiting for the data to be revised, thus the execution priority can be adjusted accordingly.

4. Deadline of a transaction

Transaction that is close to its deadline should receive a higher priority than those that are unlikely to be tardy. This is because the company may be penalized for breaching agreements if orders are not fulfilled by the deadlines. The cost associated with deadline depends on a set of transactions that are waiting to access a common set of files. To ensure that transactions have sufficient time to be completed, the slack time st of mature transactions is computed as:

$$st = dt - pt - ct$$

where ct is the current time, d is the deadline and pt is the estimated processing time of a transaction. For pending transactions, which are waiting for the files that are locked by the transactions in execution, the slack time is computed as:

$$st = dt - pt - lt$$

where lt is the latest start time of the transactions and equals to the latest completion time of transaction which locks the files required by the pending transaction. The slack time of pending transactions is computed after the slack time of mature transactions were calculated. The priority of the mature transactions could be changed if the execution of these transactions leads to an overdue of some pending transactions, given that such changes do not violate the order of works. The system first computes the slack times of mature transactions. The slack time of pending transactions is then calculated based on the completion time of all the

executing transactions that hold the required files. A transaction having a negative slack time will be tardy and the deadline-related cost will affect the schedule. Figure 8.2 illustrates the influence of processing mature transactions to pending transactions. Suppose that there are two mature transactions T_{m1} and T_{m2} and one pending transaction T_{p1} in a DPDM system. Both mature and pending transactions require a common set of files for process. The attributes of the transactions are listed in the figure. Some files that T_{p1} requires are locked by other executing transactions and the latest completion time of which is 1 time unit. However, other files which T_{p1} needs are requested by T_{m1} and T_{m2} , transactions which will become mature when they gain the access. The latest completion time of the two mature transactions is 3 units and this will be the expected start time of T_p if the system decides to process the mature transactions and subsequently T_p cannot be completed by its deadline.

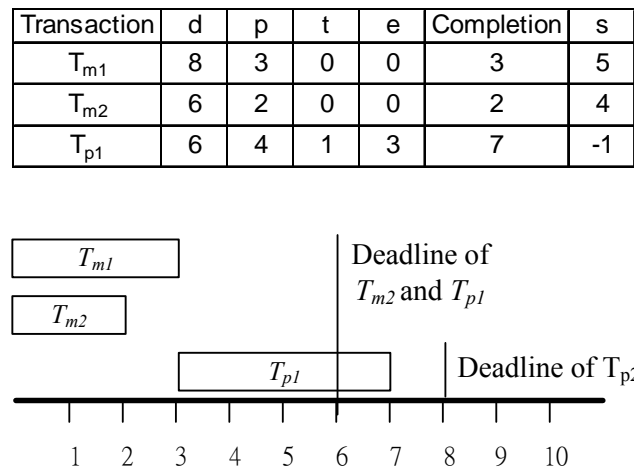


Figure 8.2 Outcome of executing transactions without scheduling

8.2 Set Partitioning Problem

This section describes the formulation of Integer Programming (IP) to model the transaction scheduling problem. Scheduling problems are often large and difficult to solve, such problems can be solved using the optimization techniques that are based on Linear Programming formulations [Ryan & Foster

1981] or Lagrangian relaxation [Ceria, et al. 1998]. These techniques are usually rigorous and provide a guarantee of optimality. The problems formulated as linear programs are commonly solved using the revised simplex method.

The following matrix-vector notation is used throughout this chapter and basic understanding of the revised simplex method for linear programming can be found in [Salkin 1989].

Set Partitioning Problem (SPP) is one particular integer linear program that is used extensively in scheduling. Given:

1. $I = \{1, \dots, m\}$
2. A collection of subsets $P = \{P_1, P_2, \dots, P_n\}$, where each $P_j \subset P$
3. A cost function $c(P_j)$

Then, some subsets from $J \subset \{1, \dots, n\}$ define a partition of I where,

1. $\bigcup_{j \in J} P_j = I$
2. $P_j \cap P_k = \emptyset$ for all $j, k \in \{1, \dots, n\}, j \neq k$

The set partitioning problem tries to seek a minimum cost partition:

$$\begin{array}{ll} \min & \sum_{j \in J} c(P_j) \\ \text{s.t.} & J \text{ partitions } I \end{array}$$

The set partitioning problem has an alternative integer linear program formulation:

$$\begin{aligned}
 & \text{minimise} && c^T x \\
 & \text{subject to:} && Ax = e \\
 & && x_j \in \{0,1\} \quad \forall j \\
 & \text{where} && \\
 & && a_{ij} = \begin{cases} 1 & \text{if } p_j \text{ contains elements } j \\ 0 & \text{otherwise} \end{cases} \\
 & && x_j = \begin{cases} 1 & \text{if } p_j \text{ is in the partition} \\ 0 & \text{otherwise} \end{cases}
 \end{aligned} \tag{8.1}$$

where e is an appropriately sized column vector of 1's.

The SPP is the basis of many large-scale scheduling problems, where each element of I can be considered as data object required, and the sets p_j represent the transactions to be processed. The solution to SPP is the set of schedules that performs each transaction exactly once and has the minimum cost. The SPP is often formulated as a set covering problem for scheduling problems by replacing the equality constraints with inequality restriction, it implies that the data objects can be accessed concurrently by more than one transactions.

8.2.1 Formulation of Concurrent Transaction Scheduling Problem

An IP can be formulated as a set covering problem by considering it as a problem of choosing a subset of transactions to process with the lowest possible cost that meets the constraints of concurrent access of files in which some data can be accessed by exactly one transaction at a time and some can be accessed concurrently by more than one transactions. The modes of locks which transactions apply to data objects create the constraints of the Concurrent Transaction Scheduling Problem (CTSP). Thus, there are five

constraints for each of the data objects in the database, one type of lock mode create a constraint of the CTSP. The constraint is described as follows:

Let K be the number of transactions waiting to be executed

M be the number of lock modes

D be the number of data objects

$$a_{dmT} = \begin{cases} 1 & \text{if transaction } T \text{ applies mode of lock } m \text{ on data object } d \\ 0 & \text{otherwise} \end{cases}$$

$$x_T = \begin{cases} 1 & \text{if transaction } T \text{ is processed} \\ 0 & \text{otherwise} \end{cases}$$

Let b_{dm} be the number of concurrent accesses to data object d for each mode of lock m , the column vector a_{dmT} is referred to as the request of lock mode m on d by transaction T , thus the concurrent access constraints are defined.

$$\left. \begin{aligned} \sum_{T=1}^K a_{dmT} x_T &\leq M y_{d_x} & m = \{S, V\} \\ \sum_{T=1}^K a_{dVT} x_T + y_{d_v} + y_{d_x} &\leq 1 \\ \sum_{T=1}^K a_{dXT} x_T + y_{d_v} + y_{d_x} &\leq 1 \end{aligned} \right\} \forall d \in PD \quad (8.2)$$

$$\left. \begin{aligned} \sum_{T=1}^K a_{dmT} x_T &\leq M y_{d_x} & m = \{IS, IV\} \\ \sum_{T=1}^K a_{d(IV)T} x_T + y_{d_{IV}} + y_{d_x} &\leq 1 \\ \sum_{T=1}^K a_{dXT} x_T + y_{d_{IV}} + y_{d_x} &\leq 1 \end{aligned} \right\} \forall d \in AD \quad (8.3)$$

The first three constraints restrict the mode of locks applied to data objects that belong to a part. The first constraint allows multiple shared accesses to a data object if it is not exclusively locked, since V mode lock on a part data object implies that a new version of the part is being created while the current version is still readable by other users. The second and third constraints require that the data object can either be locked in V mode or X mode. In other words, one of these constraints need not be satisfied so long as the other one is. Since both constraints need to be included in the problem formulation, auxiliary binary variable y and a large number M are introduced in the model to accommodate the fact that that not both need to be satisfied together but at least one of the constraints must hold. The constraints of data access can be written in the above mentioned form.

The large number M acts as infinity so it must be chosen to be larger than any possible constraint value. Since y is either 0 or 1, we will have one constraint maintaining as zero so that the solution cannot be both allowing some transactions to apply shared locks and an exclusive lock to a data object simultaneously. Consider that there are two transactions T_1 and T_2 that want to read the data object d . Each of the transactions applies a lock in shared mode to the data object. Also, suppose there is another transaction T_3 is going to modify the data object and apply an X lock to d . The equivalent problem is given as follows:

$$\begin{aligned} x_1 + x_2 &\leq My_{d_x} \\ x_3 + y_{d_x} &\leq 1 \end{aligned} \tag{8.4}$$

Because there is no transaction applying any V lock to the data object, so the second constraint does not appear in the problem.

Taking the choice $y = 0$, transaction T_3 will apply an exclusive lock to data object and no read lock can be applied to d concurrently since the right hand side value of the first constraint becomes zero as a consequence of satisfying the second constraint. Conversely if $y = 1$, the first constraint is active and

does not allow T_3 to apply an exclusive lock while the second constraint restricts the value of x_3 to zero and the data objects can hold as many shared lock as possible if the first constraint has no limit.

Assume processing transaction T has a cost c_T and a cost Z incurred if the transaction is not processed. The number of unprocessed transactions can be minimized by setting Z to a very large value. The problem can be formulated as:

$$\begin{aligned}
 \min \quad & \sum_{T=1}^K c_T x_T + Z \left(K - \sum_T x_T \right) \\
 & \left. \begin{aligned}
 & \sum_{T=1}^K a_{dmT} x_T \leq M y_{d_x} \quad m = \{S, V\} \\
 & \sum_{T=1}^K a_{dVT} x_T + y_{d_v} + y_{d_x} \leq 1 \\
 & \sum_{T=1}^K a_{dXT} x_T + y_{d_v} + y_{d_x} \leq 1
 \end{aligned} \right\} \forall d \in PD \\
 & \left. \begin{aligned}
 & \sum_{T=1}^K a_{dmT} x_T \leq M y_{d_x} \quad m = \{IS, IV\} \\
 & \sum_{T=1}^K a_{d(IV)T} x_T + y_{d_{iv}} + y_{d_x} \leq 1 \\
 & \sum_{T=1}^K a_{dXT} x_T + y_{d_{iv}} + y_{d_x} \leq 1
 \end{aligned} \right\} \forall d \in AD \\
 & x_T, y_{d_v}, y_{d_{iv}}, y_{d_x} = \{0, 1\}
 \end{aligned}$$

Figure 8.3 is an example of the above notation. In this example, transaction T_1 is trying to lock data object d_1 and d_2 in shared mode and d_3 in version mode. T_2 is going to lock d_1 in exclusive mode and d_3 in shared mode. Concurrently, T_3 tries to lock d_1 and d_4 in shared mode and exclusive mode respectively. There are a few of solutions to this problem, the first solution is to process T_1 and T_3 concurrently and to run T_2 when both transactions complete their tasks later. Executing T_1 and T_3 will force y_{IX} to be 1 and the constraint of exclusive

lock on d_{1X} is satisfied, T_2 would therefore not include in the solution in this scenario.

Alternatively, T_2 can be processed first and leave both T_1 and T_3 to later time. According to the mutually exclusive rule (6.1), the serializability will be violated if T_2 is executed concurrently with either T_1 or T_3 , because T_2 is going to modify d_1 , T_1 and T_3 require its content for their tasks, both of these transactions will have the risk of using incorrect content.

		T_1	T_2	T_3	y_{1V}	y_{1X}	y_{2V}	y_{2X}	y_{3V}	y_{3X}	y_{4IV}	y_{4X}	y_{5IV}	y_{5X}
	Transaction													
min	Cost	12	5	7										
Part	d_{1S}	1		1										
	d_{1V}				1	1								
	d_{1X}		1		1	1								
	d_{2S}	1												
	d_{2V}						1	1						
	d_{2X}						1	1						
	d_{3S}			1										
	d_{3V}	1							1	1				
	d_{3X}								1	1				
Assembly	d_{4IS}													
	d_{4IV}													
	d_{4X}			1							1	1		
	d_{5IS}													
	d_{5IV}										1	1		
	d_{5X}										1	1		

\leq

$\begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \end{bmatrix}$

Figure 8.3 Illustration of integer programming for transaction scheduling problem

8.2.2 Solution Approach

The proposed model comprises of two phases and continues iteratively when executing transactions complete their operations and release the locks on the files. The separation of the model into two phases facilitates alternations of methods used in future. Figure 8.4 shows the process of finding a transaction schedule in DPDM system. The activities in each stage of the process are as followed:

Construction phase: Evaluate the effect of executing a transaction to the DPDM system. A column is constructed to represent the demand of data object for each of the transactions and a cost that reflects the effect of executing a transaction is assigned the column.

1. User request: When a user decides to access the PDM system, a new transaction is initiated.
2. Specifying actions: the actions to be performed are specified by the user in this stage. Actions are selected from the set of predefined operations. A finite number of actions can be performed by the user upon the data. Namely, the actions are creating new data or new versions, viewing, modifying, and obsoleting existing data.
3. Action interpretation: The system will interpret the actions and perform appropriate locking operations on the data according to the predefined rules.
4. Transaction generation: An appropriate sized binary column vector will be created for each user request. The value of an element is 1 if the user needs the particular data object for the task and 0 means otherwise.
5. Calculating cost of transaction: After constructing the column, a score will be given to reflect its priority. The maturity of the files required, the type of operations to be acting on the files and the deadline of the tasks are the factors considered in the cost calculation. The procedure will be detailed in later section.

6. Transaction base: All pending transactions are stored here. It also maintains the record of transaction-in-progress. It will be activated when there is a new transaction arrived and when there is a transaction completed, as some immature transactions become ready to be executed. A transaction batch, which includes all the pending transactions and transaction-in-progress, will be passed to the transaction scheduler when it is called.

Solution phase: Solve the integer programming problem defined by the transactions that are ready to be executed using an optimization engine and update the essential information for the construction phase of this solution approach.

7. Transaction scheduler: After receiving transaction batch, it will compute a schedule that includes a subset of transactions to be executed. The selection of transactions is restricted by the access control, such that any conflicting multiple accesses to files are ruled out. The objective of the schedule is to minimize the number of tasks missing their deadlines. Unexecuted transactions will return to the transaction base and wait until required files are available.
8. Locking affected data: Specific lock will be applied on the data files according to the actions. For modification, the user who issues the transaction will be granted the ownership of the files. Only the owner can check in a modified version of that data files.
9. Transaction completed: All data locked for the transaction are disengaged. The transaction is removed from the transaction base. In turn, the base will compile a new transaction batch for creating a new schedule.

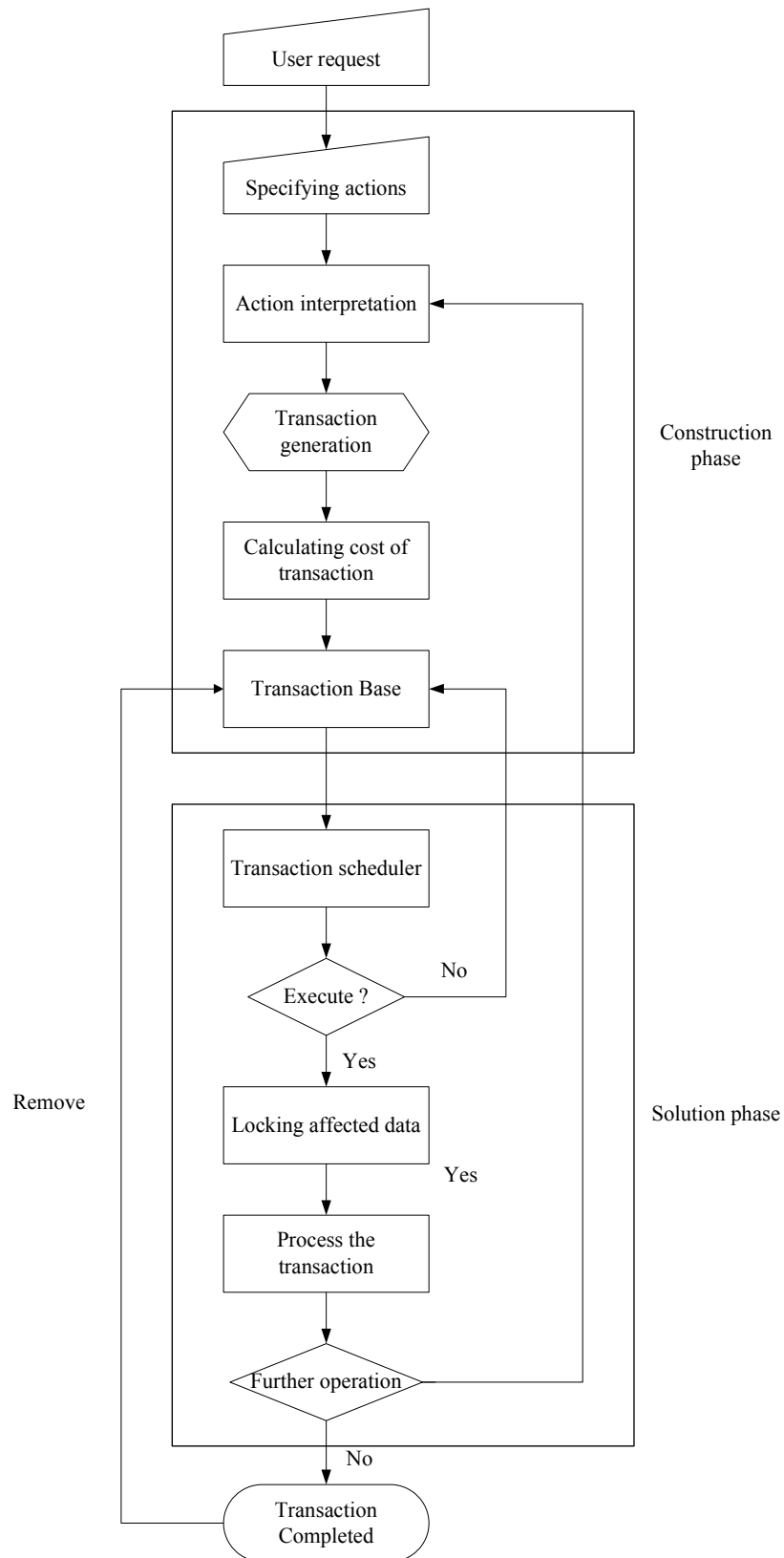


Figure 8.4 Proposed model of transaction scheduling for DPDM systems

Chapter 9

Simulations and Performance Evaluation

The algorithm presented in the previous chapters, of which was evaluated in a number of simulations. Four concurrency models simulated in this study are listed below, the procedures and results of these simulations are discussed in later sections.

1. Basic two-phase locking with First-In-First-Out (FIFO) policy.
2. Granularity version locking with First-In-First-Out (FIFO) policy.
3. Basic two-phase locking with transaction scheduling.
4. Combining granularity version locking with transaction scheduling.

9.1 General Information of the Simulation

The idea of having more concurrent accesses to the data of a PDM system is to have more tasks completed on time. In the simulations, the measure of performances of the models is the number of transactions that completed after their deadline, that is, the late transaction. The deadline of each transaction is defined as:

$$\text{Deadline} = \text{Arrival Time} + \text{Processing Time}$$

That is, if a transaction cannot start immediately when it arrived, the transaction will be counted as a late transaction.

The exponential distribution has been used to model inter-arrival times when arrivals are completely random and to model service times that are highly variable. The arrival time of a transaction is generated by adding a randomly-

generated inter-arrival time between transactions to the arrival time of the last transaction. The inter-arrival time is an exponentially distributed variable with a parameter λ , which is the number of arrivals per unit time. The processing time of a transaction is the time for the transaction to complete its task, and implies that the data objects required for the task are locked during the process. The processing time is also assumed to be exponentially distributed. The number of data objects required by a transaction is set as a random variable from a normal distribution.

9.1.1 Event Oriented Simulation

The simulation process begins by generating a set of transactions to be executed. Each of the transactions is given the following attributes: arrival time, processing time, number of data objects, and a list of data objects required to process. Each of these items is generated from a statistical distribution that best describes the phenomenon.

After a set of transactions were generated, the simulated system starts processing the data request of transactions. Suppose transaction T_1 has already locked data object d_1 and d_3 , and there are two new transactions T_2 and T_3 presented in the system with their attributes listed in the Table 9.1.

Transaction	Arrival Time	Processing time	Deadline	Data Object Required
1	2	8	12	1, 3
2	8	12	22	2, 3
3	10	2	14	2

Table 9.1 Attributes of transactions in system simulation example

The simulation process of the above example is given and illustrated in Table 9.2. The simulation time is advanced to time 2 as it is the arrival time of T_1 . T_2 arrives at time 8 before the completion of T_1 , so the simulation time is incremented to time 8. However, T_2 cannot start its process because one of its required data object, namely d_3 , is locked by T_1 . Thus, the simulation clock is next advanced to time 10, which is the time of T_3 arrival. T_2 cannot start immediately as T_3 has locked d_2 at time 8. The completion time of T_2 is calculated when it locked d_2 as the simulation knows that T_1 will release the lock on d_3 at time 11 and the processing time of T_3 is 12. Therefore, the simulation clock jumps to time 23 in which T_3 starts the process and completes at time 25, which is later than its deadline.

Clock	Transaction	Event	Data Object		
			1	2	3
0					
2	1	T_1 arrives T_1 locks d_1, d_3	(1,11)		(1,11)
8	1,2	T_2 arrives T_2 locks d_2	(1,11)	(2,23)	(1,11)
10	1,2,3	T_3 arrives	(1,11)	(2,23)	(1,11)
11	2,3	T_1 unlocks d_1, d_3 T_2 locks d_3		(2,23)	(2,23)
23	3	T_2 unlocks d_2, d_3 T_3 locks d_2		(3,25)	
25		T_3 unlocks d_2			

Table 9.2 Example of event oriented simulation on PDM system

9.2 Simulations and Results of Various Models

The procedures and result analysis of the four concurrency control models simulations are discussed in this section. All models were simulated to process 10000 transactions. There are 1000 data objects in the test, of which every 50th

data objects are considered as objects of level 1 assembly, every 20th objects are considered as of level 0 assembly and others are considered as at part level. The composition of data objects in the simulation is illustrated on Figure 9.1.

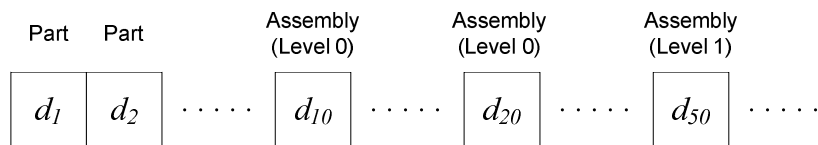


Figure 9.1 Composition of data objects in database

The arrival of transactions in these simulations follows an exponential distribution with inter-arrival rate of 0.01. The processing time of each transaction follows the same distribution with a mean processing time of 100 time units, and 1000 replications are simulated on all the models. The system is assumed that 80% of the accesses are read operations and 20% of which are write operations. The type of access operation to each data object is randomly assigned according to the proportions have just mentioned. The first model simulated is the two-phase locking with FIFO policy, which is named as the basic model in the study. Then, other concurrency control models will be simulated with the same sets of transactions and their results will be compared with the basic model using paired samples t-tests to evaluate their performance.

9.2.1 Two Phase Locking (2PL) Model

The procedure of simulating the two phase locking model of chapter 6 is described in this section. In this model, transactions start locking their required files which are free of locks once they are arrived, the deadlock avoidance implemented in this model is Wound-Wait method, where a younger transaction can wait for the data object until an older transaction releases the lock, otherwise, if the requesting transaction is older than the transaction that locks the data object, the younger transaction is rolled back and restarted to prevent occurrence of deadlock. If a transaction requests a data objects that is

at level 0 assembly, it requires locking all the parts of which the data object is composed. Likewise, if the data object requested is at assembly level, the transaction requires holding locks on all the object's components. Considering that there are two transactions shown in Table 9.3 to be processed using two-phase locking in FIFO policy. The illustration of the simulation process is shown in Table 9.4.

Transaction	Arrival Time	Processing time	Deadline	Data Object Required
1	1	8	12	1, 2
2	5	12	22	2, 3

Table 9.3 Transaction examples of event orientated simulation of PDM system

In the simulation, transaction T_1 arrives and applies a read locks on data objects d_1 and write lock on d_2 at time 1. The system updates the status of these data objects and indicates that they will be freed on time 9. The system clock is then incremented to time 5 in which the next event occurs, that is, T_2 arrives and applies a write lock on d_3 and waits for d_2 , as write lock is mutually exclusive to other locks. Suppose T_1 requires the content of d_3 for its task in time 6, under the rule of Wound-Wait method, T_2 is restarted and unlocks d_3 , because it is younger than T_1 . Therefore, T_1 locks d_3 at time 6 and T_2 waits for both d_2 and d_3 until time 9.

Clock	Transaction	Event	Data Object		
			1	2	3
0					
1	1	T_1 arrives T_1 locks d_1, d_3	R(1,9)	W(1,9)	
5	1, 2	T_2 arrives T_2 locks d_3 T_2 waits for d_2	R(1,9)	W(1,9)	W(2,17)
6	1, 2	T_1 requests d_3 restart T_2 T_1 locks d_3	R(1,9)	W(1,9)	W(1,9)
9	2	T_1 unlocks d_1, d_2, d_3 T_2 locks d_2, d_3		R(2,21)	W(2,21)
21		T_2 unlocks d_2, d_3			

Table 9.4 Simulation of two-phase locking with Wound-Wait and FIFO policy

Simulation Result

A sample mean of late transactions is calculated to analyse the simulation results of the basic two phase locking model. The model was simulated with 10000 transactions in each replication and the number of late transactions was observed at the end of the process. The sample mean of late transactions over 1000 replications is shown in Table 9.5. The average number of late transactions of this concurrency control model is 189.50. The performance of other concurrency models described in later sections will be evaluated by comparing their simulation results with this result.

	N	Mean	Std. Deviation	Std. Error Mean
FIFO_2PL	1000	189.50	351.566	11.117

Table 9.5 Average late transactions in the basic model

The histogram in Figure 9.2 is based on the number of late transactions in 10000 transactions over 1000 replications. The start interval starts at 0 and interval width is 50. In the evaluation the basic model, 379 replicas have 0-25 late transaction, 185 replicas have 26-75 late transactions, and 96 replicas have 76-125 late transactions. The replicas of these three intervals constitute more than 66% of all observations. Notice that there are some replications having a very large number of late transactions. The worst case of the simulation in the basic model has 3488 late transactions.

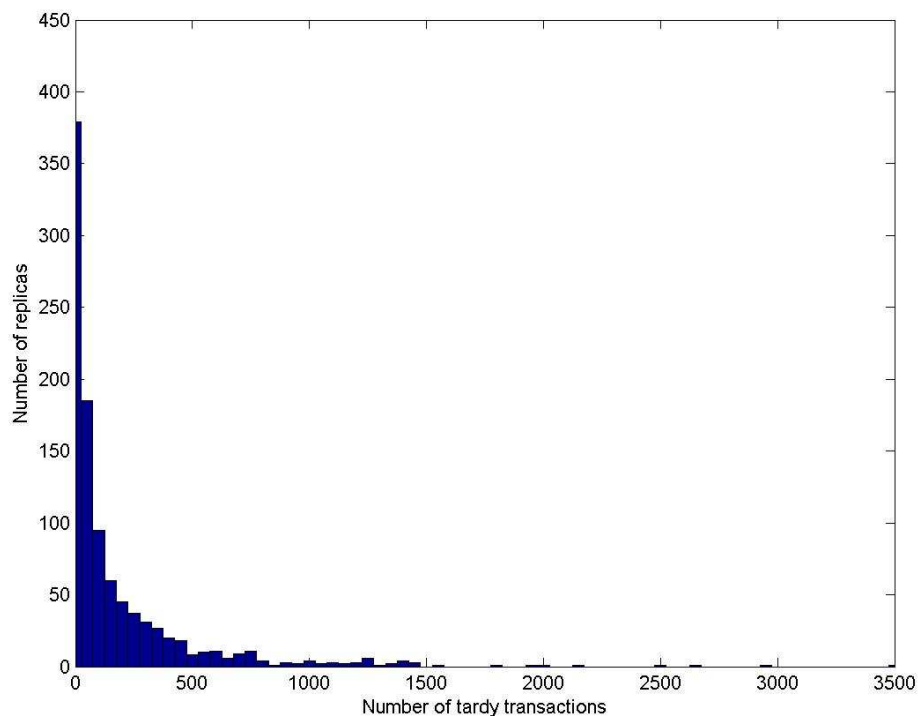


Figure 9.2 Histogram of the late transactions in the basic model

9.2.2 Granularity Version Locking

In the simulation of this model, transactions lock the data objects using the proposed method described in Chapter 7. The model also adopted the Wound-Wait method as its deadlock avoidance. The example transactions to be processed and the simulation of the model are shown in Tables 9.6 and 9.7 respectively.

In the simulation example of the granularity version locking, there are two transactions to be processed. The task of T_1 is to modify a level 0 assembly, which is the data object d_{10} in the system. It also requires the content of d_9 to be processed. Therefore, T_1 locks d_9 and d_{10} in Shared mode and Intent Version mode respectively in time 1. In time 2, T_1 begins modifying d_{10} , and a new version of the data object is created. A new data object, $d_{10.2}$, is immediately locked in exclusive mode to prevent any concurrent access from other transactions. The simulation clock then forwards to time 4 at which T_2 arrives and locks the required data object. T_2 needs to refer to the content of d_{10} for modifying d_{18} , thus it locks $d_{10.1}$ in Intent Shared mode. Also, T_2 must lock the direct assembly of d_{18} in Intent Version mode and lock d_{18} in Version mode before modification. In the next time unit, a new version of d_{18} is created by T_2 and is locked in exclusive mode until the process completes. Since there is no other events occur before T_2 completes, the clock jumps to time 14 at which T_2 finishes modifying d_{18} and releases all its locks. T_1 completes its task and releases the locks on d_9 , $d_{10.1}$, and $d_{10.2}$ in time 21.

Arrival Time	Processing time	Deadline	Data Object Required
1	20	25	9, 10
4	10	16	18, 20

Table 9.6 Transactions in granularity version locking simulation

Clock	Transaction	Event	9	10.1	18.1	20	10.2	18.2
0								
1	1	T_1 arrives T_1 locks d_9, d_{10}	S(1,21)	IV(1,21)				
2	1	T_1 modifies d_{10}	S(1,21)	IV(1,21)			X(1,21)	
4	1, 2	T_2 arrives T_2 locks d_{10}, d_{18}, d_{20}	S(1,21)	IV(1,21) IS(2,14)	V(2,14)	IV(2,21)	X(1,21)	
5	1, 2	T_2 modifies d_{18}	S(1,21)	IV(1,21) IS(2,14)	V(2,14)	IV(2,21)	X(1,21)	X(2,14)
14	1	T_2 unlocks d_{10}, d_{18}, d_{20}	S(1,21)	IV(1,21)			X(1,21)	
21		T_1 unlocks d_9, d_{10}						

Table 9.7 Simulation example of granularity version locking in FIFO

Evaluation of Granularity Version Locking

The granularity version locking (GVL) model was simulated with 10000 transactions in each replication and the number of late transactions was observed at the end of the process. The resulting frequency distribution is displayed in Figure 9.3. The left-skewed histogram indicates that most of the replications have fewer than 500 late transactions. There are 403 replicas have 0-25 late transaction, 154 replicas have 26-75 late transactions, and 94 replicas have 76-125 late transactions. There are a few replicas having large number of late transactions and the worst result in GVL model has 2259 late transactions.

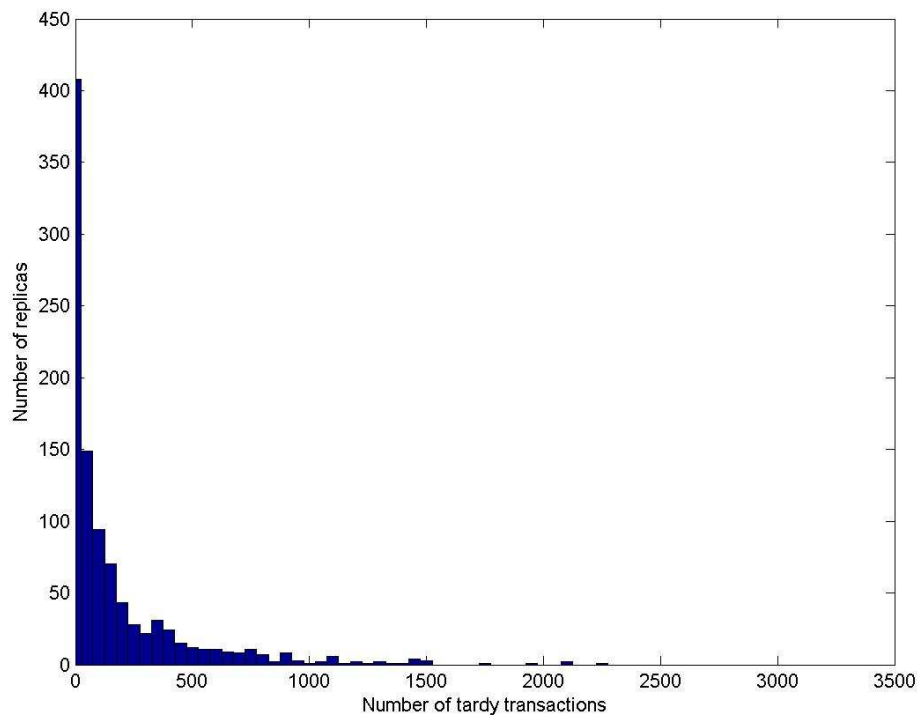


Figure 9.3 Histogram of the tardy transactions in the granularity version locking model

The sample means of late transactions over 1000 replications of the basic model and granularity version model are shown in Table 9.8. FIFO_2PL is the result of the two phase locking model and has average late transactions of 189.50 in 1000 replications. FIFO_GVL is the result of the GVL in Wound-

Wait deadlock avoidance and FIFO queuing discipline and its average late transactions is 164.73.

Paired Samples Statistics

	Mean	N	Std. Deviation	Std. Error Mean
FIFO_2PL	189.50	1000	351.566	11.117
FIFO_GVL	164.73	1000	304.554	9.631

Table 9.8 Mean of late transactions of granularity version locking

Paired sample t-test is done to compare the average number of late transactions in the two models to ensure that the observed differences are due to difference in performances of the designs. The null hypothesis here would be that the average late transactions are same under the concurrency control of the basic model and the granularity version model.

The purpose of this statistical analysis is to validate that the granularity version locking performs better than the basic two phase locking. In Table 9.9, the mean, standard deviations and standard error of mean of these differences are calculated by the paired value of each replication obtained from FIFO_2Pl and FIFO_GVL. The two-tailed p-value is 0.000, which is less than the conventional 5% or 1% level of significance. Therefore, the null hypothesis can be rejected and the average late transactions in the basic model are indeed more than the granularity version model. Owing to the fact that the difference between the mean late transactions of the basic model and the proposed locking model ($189.50 - 164.73 = 24.77$) is totally to the right of zero, then there is a strong evidence that the basic model has more late transactions than the proposed locking model. This implies that the performance of granularity version model is better than the basic model.

Paired Samples Test

	Paired Differences					t	df	Sig. (2-tailed)
	Mean	Std. Deviation	Std. Error Mean	95% Confidence Interval of the Difference				
				Lower	Upper			
FIFO_2PL – FIFO_GVL	24.77	57.002	1.803	21.23	28.31	13.741	999	.000

Table 9.9 Summary of measures for comparing the basic model and the granularity version locking model

9.2.3 Transaction Scheduling

The simulation of the proposed transaction scheduling method with the basic two phase locking concurrency control of chapter 8 is described in this section. The model can be formulated as a set covering problem by considering the following list of constraints to ensuring the data integrity of a PDM system:

1. A write lock is applied to a data object that will be modified.
2. No lock can be applied to data objects that hold an exclusive lock.
3. Write lock cannot be applied to files that hold any locks.
4. Multiple read locks can be applied to files.

In two phase locking, data objects that are read locked allow multiple read-only accesses; the other constraints in this IP are accesses that determined by write locks. Transactions that are trying to modify read locked data objects and to read data objects that are write-locked will be excluded from the scheduling process, because these transactions will not be processed anyway due to the violation of the constraints. Thus, mature and pending transactions that require only lock-free data objects will be included. Access conflicts occur when some transactions are requesting a write lock to a data object, and other transactions are trying to apply read locks to it. Using notations described in earlier sections of this chapter, the restrictions to prevent the occurrence of conflicts form the constraints of the model as below, where the column vector a represents the

request of two lock modes, R (Read) and W (Write) on data object d by transaction T .

$$\begin{aligned}
 \min \quad & \sum_{T=1}^K c_T x_T + Z \left(K - \sum_{T=1}^K x_T \right) \\
 \text{s.t.} \quad & \sum_T a_{dWT} x_T \leq 1 - y_d \\
 & \sum_T a_{dRT} x_T \leq M y_d \\
 & x_T, y_d = \{0, 1\}
 \end{aligned} \tag{9.1}$$

Transactions in a PDM system require a set of data objects to complete a task. They are represented as columns in the IP matrix in the model. There are two constraints on each of the data objects in the system. The first constraint in the model guarantees that data objects hold at most one write lock. The second constraint allows a data object an unlimited number of read locks. These constraints are mutually exclusive alternatives. Noted that in this simulation model, all data objects of parts of which the assembly level data object is subjected to modification are also locked in exclusive mode by the requesting transaction. A simulation example of this model is presented.

Considering that the three transactions listed in Table 9.10 are to be executed. Their simulated executions in this model are illustrated in Table 9.11. At time 1 on the simulation clock, transaction T_1 arrives and locks the data object d_{10} for modification, since T_1 intends to modify a level 0 assembly object, it has to write-lock all the data objects of which d_{10} is composed, that is, d_{1-10} are all locked in write mode from time 1 until the task of T_1 completes. During the execution of T_1 , T_2 , and T_3 arrive at time 4 and 5 respectively. None of both can be executed immediately upon their arrivals as some of the data objects they require are locked in write mode by T_1 . In the creation of columns for T_2 and T_3 , which is step 5 of the transaction scheduling model shown in Figure 8.5, both of these transactions are passed to the transaction base. However the transaction scheduler is not evoked. This is because the model notices that both

transactions cannot be executed under the definition of mutual exclusion (6.1). At time 6, T_1 completes the task and releases the data objects it has locked. Since T_2 requires reading d_7 and T_3 decides to modify it. The transaction scheduler is triggered to select which transaction is to be processed. The schedule is determined by finding a solution with the minimal processing cost. One of the components of a transaction cost is related to its processing time and deadline. In this case T_3 is chosen to be processed in time, because it has a tighter deadline than T_2 . Thus, T_3 locks d_7 , d_8 , and d_9 . The clock jumps to time 16 at which T_3 finishes its task and releases the locks on d_7 . Afterward, T_2 locks its requested data objects and completes its task at time 25.

Transaction	Arrival Time	Processing time	Deadline	Data Object Required
1	1	6	10	10
2	4	9	25	7, 16, 17
3	5	10	16	7, 8, 9

Table 9.10 Three transactions in the simulation of transaction scheduling method

Clock	Transaction	Event	Data Object					
			7	8	9	10	16	17
0								
1	1	T_1 arrives T_1 locks d_{1-10}	W(1,6)	W(1,6)	W(1,6)	W(1,6)		
4	1, 2	T_2 arrives	W(1,6)	W(1,6)	W(1,6)	W(1,6)		
5	1, 2, 3	T_3 arrives	W(1,6)	W(1,6)	W(1,6)	W(1,6)		
6	2, 3	T_1 unlocks d_{1-10} T_3 locks d_7, d_8, d_9	R(3,16)	R(3,16)	R(3,16)			
16	2	T_3 unlocks d_7, d_8, d_9 T_2 locks d_7, d_{16}, d_{17}	W(2,25)				R(2, 25)	R(2, 25)
25		T_2 unlocks d_7, d_{16}, d_{17}						

Table 9.11 Simulation example of transaction scheduling method

Evaluation of Transaction Scheduling

The transaction scheduling model was simulated with 10000 transactions in each replication and the number of late transactions was observed at the end of the process. The frequency distribution in Figure 9.4 shows that there are 395 replicas in the simulation have 0-25 late transaction, 189 replicas have 26-75 late transactions, and 91 replicas have 76-125 late transactions. There are a few replicas having large number of late transactions and the worst result in TS model has 2941 late transactions.

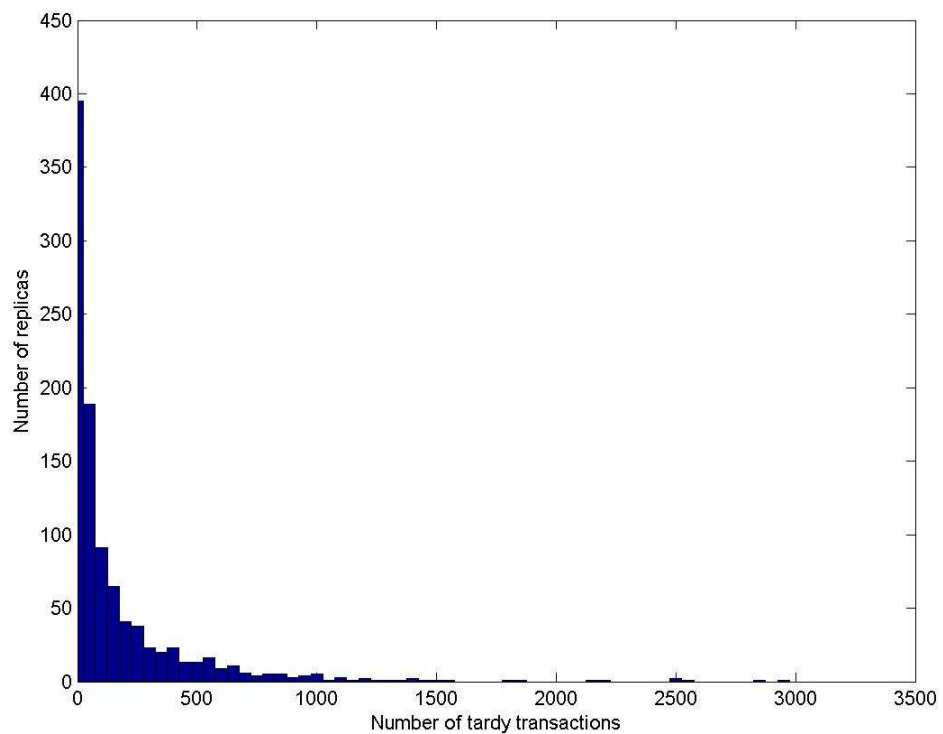


Figure 9.4 Histogram of the tardy transactions in the transaction scheduling model

The sample means of tardy transactions over 1000 replications of the basic model and transaction scheduling model are shown in Table 9.12. TS_2PL is the simulation result of the transaction scheduling model with two phase locking concurrency control and the average tardy transactions of this model are 177.10.

Paired Samples Statistics

	Mean	N	Std. Deviation	Std. Error Mean
FIFO_2PL	189.50	1000	351.566	11.117
TS_2PL	177.10	1000	301.348	9.529

Table 9.12 Mean tardy transactions of transaction scheduling model

Because the observations of the tardy transactions contain random variation, in order to validate that the observed differences are not due to the random fluctuation inherent in the models, paired sample t-test is done to compare the average number of tardy transactions in the two models. The null hypothesis here would be that the average tardy transactions are same under the execution of transactions in FIFO queuing discipline and in scheduling model. The test summary is shown in Table 9.13.

Paired Samples Test

	Paired Differences					t	df	Sig. (2-tailed)
	Mean	Std. Deviation	Std. Error Mean	95% Confidence Interval of the Difference				
				Lower	Upper			
FIFO_2PL - TS_2PL	12.40	81.505	2.577	7.35	17.46	4.813	999	.000

Table 9.13 Summary of measures for comparing the basic model and the transaction scheduling model

The analysis summary shows that the two-tailed p-value is 0.000, which is less than the conventional 5% or 1% level of significance. Therefore, the null hypothesis can be rejected and the average tardy transactions in the basic model are different to the transaction scheduling model. Given that the confidence interval for the difference of the mean tardy transactions of the basic model and the transaction scheduling model ($7.35 < \text{FIFO_2PL} - \text{TS_2PL} < 17.46$) lies completely above zero and so provides strong evidence that the basic model certainly has more tardy transactions than the transaction scheduling model, that is, the performance of the transaction model is better than the basic model.

9.2.4 The Combined Model

The simulation of a model combining the granularity version locking and transaction scheduling deadlock avoidance method is described in this section. The simulation of the combined model described in Section 8.2.1 is illustrated using the transaction example in Table 9.10. The simulation of the execution of these transactions is shown in Table 9.14. At time 1 of the simulation clock, transaction T_1 locks the data object d_{10} in Intent Version mode for modification. T_1 has to lock all the data objects of which d_{10} is composed, $d_{1..9}$, in Intent Versions mode. At time 2, T_1 starts modifying d_{10} and creates the second version of d_{10} . The simulation clock then forwards to time 4 at which T_2 arrives and tries to lock the required data object. However, the transaction scheduler is not evoked as the data object, d_7 , that T_2 intends to modify has already been locked in Version mode by T_1 . The rule of granularity version locking does not allow concurrent V locks applied on the same data object. The simulation time is then incremented to time 5, T_3 arrives and locks $d_{7..9}$ in Shared mode as locks of Shared mode and Version are compatible. At time 6, T_1 completes the task and releases all the data objects it has locked; the release event evokes the transaction scheduler to determine new transaction execution. Subsequently, T_2 locks d_7 in Version mode and d_8 and d_9 in Shared mode. In the next time increment, T_2 modifies d_7 and a new version of the data object is created. Since there is no event happens between times 8 and 14, the simulation clock jumps to time 15 at which T_2 completes the task and releases all the locks.

Clock	Transaction	Event	Data Object								
			7.1	7.2	8	9	10.1	10.2	16	17	
0											
1	1	T_1 arrives T_1 locks d_{1-10}	V(1,6)		V(1,6)	V(1,6)	IV(1,6)				
2	1	T_1 modifies d_{10}	V(1,6)		V(1,6)	V(1,6)	IV(1,6)	X(1,6)			
4	1, 2	T_2 arrives	V(1,6)		V(1,6)	V(1,6)	IV(1,6)	X(1,6)			
5	1, 2, 3	T_3 arrives T_3 locks d_7, d_8, d_9	V(1,6) S(3,15)		V(1,6) S(3,15)	V(1,6) S(3,15)	IV(1,6)	X(1,6)			
6	2, 3	T_1 unlocks d_{1-10} T_2 locks d_7, d_{16}, d_{17}	S(3,15) V(2,15)		S(3,15)	S(3,15)				S(2,15)	S(2,15)
7	2	T_2 modifies d_7	S(3,15) V(2,15)	X(2,15)	S(3,15)	S(3,15)				S(2,15)	S(2,15)
15	2	T_3 unlocks d_7, d_8, d_9 T_2 unlocks d_7, d_{16}, d_{17}									

Table 9.14 Simulation example of combined model

Evaluation of Combined Model

The model simulated in this section adopts the granularity version locking as concurrency control and the transaction scheduling as deadlock avoidance. The combined model is also simulated with the same 1000 replications of 10000 transactions that were used to evaluate the other models described in the previous sections. The simulation result of the combined model (TS_GVL) is plotted in Figure 9.5. The histogram in Figure 9.5 represents the frequency of replicas in the simulation of the combined model. There are 434 replicas having 0-25 late transaction, 179 replicas having 26-75 late transactions, and 75 replicas in the interval between 76-125 late transactions. This model has fewer replicas that are having large number of late transactions and the worst result in the combined model has 2177 late transactions. The mean average tardy transactions of this model are 157.97 as shown in Table 9.15.

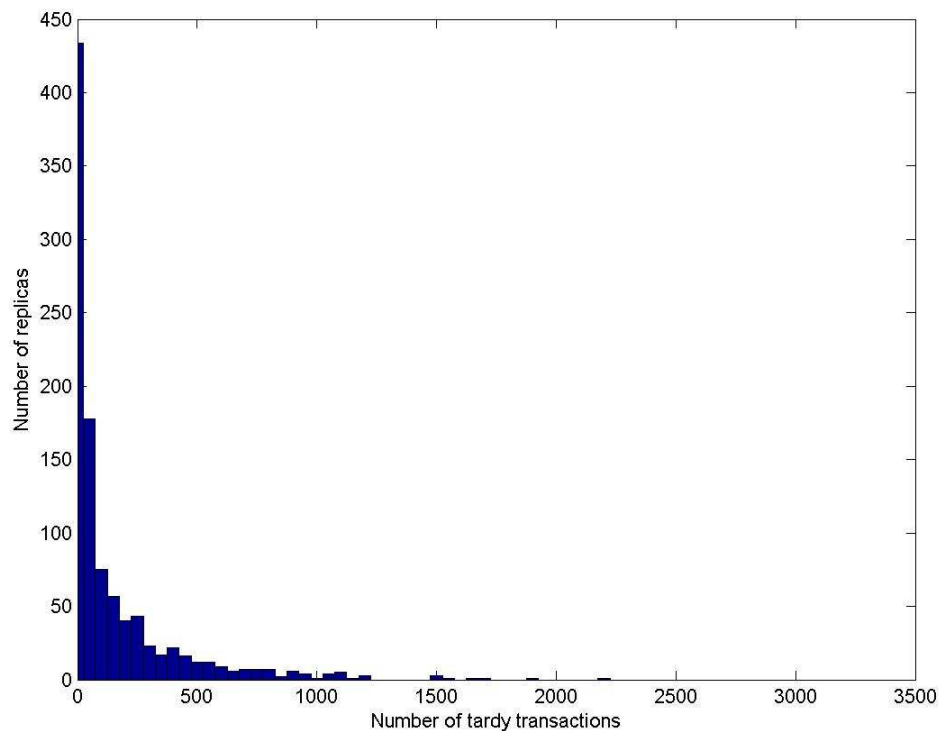


Figure 9.5 Histogram of the tardy transactions in the combined model

		Mean	N	Std. Deviation	Std. Error Mean
Pair 1	FIFO_2PL	189.50	1000	351.566	11.117
	TS_GVL	157.97	1000	288.218	9.114
Pair 2	TS_2PL	177.10	1000	301.348	9.529
	TS_GVL	157.97	1000	288.218	9.114
Pair 3	FIFO_GVL	164.73	1000	304.554	9.631
	TS_GVL	157.97	1000	288.218	9.114

Table 9.15 Mean tardy transactions of the combined model

The performance of the combined model is compared with other models using paired t-tests and the statistical summary is shown in Table 9.16. The first row of the table is the comparison of the basic model (FIFO_2PL) and the combined model (TS_GV). The difference of the mean tardy transaction of the two models is 31.53 ($189.50 - 157.97$) and the confidence interval of the difference is (27.21, 35.85). Given that the two-tailed p-value is 0.000, which is less than the conventional 5% or 1% level of significance, there is strong evidence that the performance of the two models are not the same. In addition, the confidence interval for the difference of the mean tardy transactions of these models lies completely above zero and so provides strong evidence that the basic model certainly has more tardy transactions than the transaction scheduling model. This implies that the performance of the combined model is better than the basic model.

The performance of the combined model (TS_GVL) is also compared with the granularity version locking (FIFO_GVL) and the transaction model (TS_2PL). As shown in Table 9.15, the mean tardy transactions of the combined model are fewer than models that adopt the scheduling technique or the version locking concurrency control only. The mean difference between TS_2PL and TS_GVL is 19.13 ($177.10 - 157.97$) and between FIFO_GV and TS_GVL is 6.76 ($164.73 - 157.97$). Given that p-values from respective analyses are less than the level of significance, the hypothesis of considering that there is no difference between the combined model and the other two can be rejected. Furthermore, the confidence intervals of mean differences are also well above zero for the comparison of TS_2PL - TS_GVL (15.86, 22.39) and FIFO_GVL - TS_GV (5.15, 8.38). To conclude, the combined model that integrates both

the transaction scheduling method and the granularity version locking is better than using either the transaction scheduling or the granularity version locking alone.

Paired Samples Test

		Paired Differences				t	df	Sig. (2-tailed)	
		Mean	Std. Deviation	Std. Error Mean	95% Confidence Interval of the Difference				
					Lower	Upper			
Pair 1	FIFO_2PL - TS_GVL	31.53	69.672	2.203	27.21	35.85	14.311	999	.000
Pair 2	TS_2PL - TS_GVL	19.13	52.620	1.664	15.86	22.39	11.495	999	.000
Pair 3	FIFO_GVL - TS_GVL	6.76	26.047	.824	5.15	8.38	8.210	999	.000

Table 9.16 Statistical summary of performance comparison

Chapter 10

Discussion

Issues related to the development of concurrency integrity model for DPDM system are discussed in this chapter.

10.1 DPDM System Representation

Data model of PDM system is presented in an ontological approach. A production PDM system that adopts this data model representation can be updated for new application-specific features because it can incorporate new concepts and relations. The ontology of PDM system provides a set of terms with which to describe the facts about PDM system, which include the relations between data objects and the behaviours of the functions. This study has defined some relations of components between objects and only the definitions of the necessary functions have been described for the investigation of concurrency integrity maintenance mechanism of PDM system. Also, object-oriented concept is adopted to organise and process the information according to the description of the real world objects. The model works best on object-oriented PDM system. However, most of the existing systems will need to be converted or need to interface to the object oriented systems. The design of the new systems will need to be adjusted to accommodate the needs of these legacy systems.

10.2 DPDM Specifications

The ontology provides some important semantics and relations on which the specifications of PDM functions are developed. UML sequence diagram is chosen as the diagrammatic modeling tool for the specification development. Despite that there are many graphical modeling tools available, sequence

diagrams are an intuitive visual notation for showing the time ordering of events between system components in a PDM system. However, when used for specification, sequence diagrams represent classes of objects instead of individual data objects and external system components. Also, the lack of formal semantics creates additional complexity to the validation of the specifications of PDM system model, since there is no standard on how such model can be simulated. In this study, temporal logic formulae are derived from the sequence diagrams to demonstrate the viability of using UML as a front-end for a formal notation and to overcome the UML limitations. Among a number of types of temporal logic, the UML sequence diagrams are translated from Propositional Temporal Logic (PTL) formulae. The preference for PTL is based on the balance between the expressiveness and the ease of readability and learning. The formulae in PTL can be easily modified into Timed Propositional Temporal Logic (TPTL), an extended version of PTL, when measure of time units between events is needed.

10.3 Granularity Version Locking Method

Concurrency control in PDM systems is more complex than in database system for business data management like bank account and stock trading. The main factors for this added complexity are long-duration process in product development and the existence of multiple-valid states of data objects. Engineering change is a norm rather than an exception in contemporary product development. As a result, time-varying change management and version management are the standard. Granularity version locking proposed in this study addresses these problems for concurrency control in DPDM systems and improves concurrency ability of DPDM systems by adjusting the accessibility of data objects in accordance with the action to be performed by the users and the product architecture of the physical object. The total number of lock modes in the proposed model is five. The result of the simulations shows that the locking model worked efficiently on a distributed PDM system and the integration with the transaction scheduling method successfully decreased the number of tardy transactions. Although coarse granularity and

fewer locks incur less overhead in testing, setting, and maintaining these locks, the disadvantage is that there will be less concurrency [Date 2004]. The balance between shareability and degree of concurrency has been researched for a long time [Gary, et al. 1975, Lee & Liou 1996]. There is no definite answer to the issue since the performance really depends on the characteristics, for example, system environment, data model, complexity of data etc. of a system on which the concurrency control is implemented.

10.4 Transaction Scheduling Method

Problems of keeping the PDM system in a consistent state occur when there are concurrent accesses to the data. A PDM system must provide transaction on an equivalent mechanism for concurrency control not only for regular transactions but also for long duration ones. In order to maintain the consistency of a PDM system, the transactions must process according to the concurrency control to ensure the executions are serializable. The method proposed transaction scheduling is a deadlock avoidance mechanism that is formulated as an integer programming in which the constraints represent the inherent meanings of the formulae in a concurrency control. From the simulations in Chapter 9, it shows that the method works seamlessly with the basic 2PL and the granularity version locking protocols. It ensures that there is no deadlock by controlling a transaction to lock the whole set of requested data objects, instead of allowing it to lock them once they are available. The occurrence of livelock is also eliminated because the cost of a transaction inversely proportional to its waiting time, the longer a transaction waits, the smaller its cost. Thus, the transactions will be processed eventually. The method also reduces the number of tardy transactions as a result of ordering the transactions by their deadlines and other attributes. The simulations of the transaction scheduling method assumed that the attributes of transactions do not change during the process. However, the work of the product development will be affected and changes are unavoidable. Thus, the schedule computed in each of the iterations is optimal, the overall schedule of the execution of all the known transactions may not be optimal, since the attributes of the transaction cost will be changed

by the execution of other transactions and new transactions may emerge from time to time.

Chapter 11

Conclusions

PDM commercial tools have been available in the market for nearly a decade. These tools enable enterprises to conduct its business activities in a more efficient way via ingenious management of product information. With the advent of the internet and web based technologies, PDM systems can now be executed more effectively and efficiently. The development of web-based PDM system is essential for supporting collaborative design and manufacturing at geographically dispersed sites. Heavy information flow among the design offices and production plants is anticipated everyday and its proper management and seamless integration is crucial to the success of the business. However, the contemporary commercial PDM systems cannot meet this challenge as they are ad-hoc solutions. Thus, this research focuses on bridging the gap by investigating the theoretical aspects of PDM system.

11.1 Contribution of the Research

The reported methodology is a step forward to develop a generic analysis tool for making DPDM system implementation commercial tool-independent. The major contributions of this research are shown below:

11.1.1 Data Modeling for DPDM system

In order to develop a generic representation scheme that can properly model and document enterprise-dependent workflow for current and future use, defining a PDM system using ontology seems to be a suitable approach because ontology is an explicit specification of a conceptualization. The definitions of data objects, functions, and their relations are denoted in formal

axioms that constrain the meaning. With such an ontology, existing commercial PDM systems can be described by the set of representational terms. Adoption of OO technology in PDM system is encouraged. In particular, a model that provides a guideline for implementing environmental compliance management (ECM) system by employing a PDM system and web-technology is proposed. The model includes a logic process unit that retrieves product data from PDM systems to analyse the content of hazardous materials in a product. Finally, an ontology-based ECM system for WEEE and RoHS compliance was developed and tested.

11.1.2 Graphical and Logical Representation of DPDM Specifications

The specification of PDM/DPDM system are presented using UML sequence diagram. Since UML diagrams cannot enumerate all the possible scenario arises from interactions between the systems and its users, therefore specification in formal notation is needed for verification on consistency of concurrent processes. The core PDM/DPDM functions are formulated using first order logic based upon the ontological data model. The formal language provides a computer-processable format, such that error-free checking and resource minimal analysis can be performed. This provides a sound procedure for the determination of semantic equivalences between a semi-formal notation and a formal language. Given that the product data involves real time management of resources, this research introduces temporal logic to specify the dynamic behaviour of PDM/DPDM system. Such incorporation facilitates modifications to the specification in FOL while maintaining its validity and consistency.

11.1.3 Granularity Version Locking for PDM/DPDM system

This research introduces a granularity version locking protocol for PDM/DPDM system which allows more concurrent accesses to the data by carefully selecting the type of locks being applied for the corresponding action.

The operations of this concurrency model are specified in PTL. Since temporal logic is powerful tool for describing temporal behaviour, and was chosen as the specification technique for that reason. Such that the model can be used to determine to what extent verification is possible in the future. Also, two modes of lock, Version lock (*V*) and Intent Version (*IV*), are introduced to support versioned concurrency control of a data object during its update process. *V* lock is applied to a data object of part level when a user modifies it; a new version of the data object is being created while the current version is shareable for other users. *IV* lock is applied to an assembly level data object when a data object of its parts is being modified. These two types of lock allow the current version of the data object can be viewed by other transactions but disallow modification in order to maintain the data consistency.

11.1.4 Transaction Scheduling Method for DPDM Deadlock Avoidance

A correctness criterion of distributed databases is that transactions are executed in a serializable schedule. To ensure an interleaved execution of transactions produces the same effect as a serial execution of those same transactions, conflicting transactions must not process concurrently and the order of executions is critical. Any concurrency control that bases on the strict two-phase locking protocol can be used to solve the concurrency problems. However, locking has problems of its own, in particular, the problem of deadlock.

In this research, a transaction scheduling algorithm for PDM/DPDM system is proposed to prevent the occurrence of deadlocks. An integer programming based scheduling technique is used to control transaction executions in a system. The algorithm is designed to improve the concurrency of a system by ordering the process order of transactions according to their status. This is also integratable to various concurrency control methods; such that the performance of a system can be improved without sacrificing the data consistency and it can be implemented to any existing PDM/DPDM systems.

11.2 Future Research

Based on the research in the correctness of PDM/DPDM system, future investigations are suggested on the following areas:

11.2.1 Standard Object Oriented Database Language

One of the goals of object oriented modeling is to preserve a direct correspondence between real world and database objects, such that the relationships between their components can be represented and identified easily. However, none of the commercial PDM systems is truly designed on the OO based model. They are built on the top of a relational database system and therefore cannot fully utilise the benefits of the advantages of OO data model over traditional relational data model. In order to develop an operational OOPDM system, the concern of standard semantics for OO database computation must be addressed.

11.2.2 Utilisation of the capability of an ontology based PDM system

The use of ontology in PDM system is demonstrated through the implementation of an environmental constraint compliance management system. The ontologies defined in product development can be seen as meta-data that represent semantics of the data. Ontology based reasoning service can utilise such semantics to provide a good approximation. A result from an analysis based on incomplete information may be inaccurate, which could impair the production process if the missing data leads to incompliance. By adding an estimation function to predict the probability of incompliance, user can make a decision based on the estimation computed by referencing to past projects. The efficiency of a production process can be improved since there are many stages and resources involved in manufacturing products, a company can be benefited from no halting in the product development and production flows for unavailable information.

11.2.3 Automatic Translation from UML Sequence Diagrams to Formal Semantics

UML sequence diagram is a good tool for modeling the dynamic aspects of a system. In this research, the basic functions of PDM system are represented in sequence diagrams. However, sequence diagram is not the ultimate answer to the requirement specification problems because of its semi-formality. These functions are then written as temporal logic formulas that can be reasoned and verified. Because of the complexity of temporal logic, modellers seldom use it to specify and reason systems they design. Thus, a graphical modeling tool that automatically generates models in a formal notation is valuable. To cater for these features, sequence diagram and temporal logic are good companions for the implementation of a graphical formal notation tool.

11.2.4 Relaxation of Two Phase Locking

Two phase locking protocol enforces that transactions must not request a new lock after releasing some lock in order to maintain the data consistency of a database. Eswaran [Eswaran, et al. 1976] shows that two phase restriction sometimes is not a necessary condition for consistency. For example, in a PDM system, data objects become constant after released. These data objects can no longer be modified by any transaction and by no mean they will get into an inconsistent state. Thus, a processing transaction that accesses these data objects in a non-two phase manner will not become inconsistent. And there are many situations that two phase restriction is not necessary. Therefore, it seems difficult to give nontrivial necessary conditions for a PDM system to be consistent. Further research should be conducted on two phase locking relaxation as this will help in improving the efficiency of the transaction scheduling algorithm since transactions need not be included into the scheduling process if they can lock and unlock data objects without impairing the consistency of the system.

11.2.5 Dynamic Programming for Concurrent Transaction Scheduling

The proposed transaction scheduling method is designed to eliminate the occurrence of deadlocks by ordering the execution of transactions that access some data objects concurrently according to their costs. The resultant schedule constructed by the method is composed of a set of transactions to be executed which constitute the lowest processing cost “in that instant”. However, the choice of transactions to be processed will affect the costs of the pending transactions and of the future schedules. In other words, the schedule is only a locally optimal solution. Thus, the whole schedule, which is the constitution of a series of schedules for the execution of all the known transactions, may not be the best overall solution. In order to compute a global optimal schedule, dynamic programming is a potential candidate to cope with the dynamic behaviour of the transactions. Consider the transaction scheduling problem solved at each stage, constructs a number of possible schedules, which are referred as states in dynamic programming. At each of the succeeding stages, the cost and the validity of transactions changes according to the decision in each state of the preceding stage. The schedule with the lowest cost in the final stage will be the optimal solution for the transaction scheduling problem.

References

- Abbott, R. K. & Garcia-Molina, H. (1992). Scheduling real-time transactions: A performance evaluation. *ACM Transactions on Database Systems*, 17(3), 513-560.
- Abernethy, K. & Kelly, J. C. (1992). Comparing object-oriented and data flow models - a case study. *Proceedings of the 1992 ACM annual conference on Communications*, 541-547.
- Adler, M. (1988). Algebra for data flow diagram process decomposition. *IEEE Transactions on Software Engineering*, 14(2), 169-183.
- ASME (1998). PDM for the enterprise, *Mechanical Engineering*, 120, 84: American Society of Mechanical Engineers.
- Bansler, J. P. & Havn, E. C. (2003). Building community knowledge systems: an empirical study of IT-support for sharing best practices among managers. *Knowledge and Process Management*, 10, 156-163.
- Bellini, P., Mattolini, R., & Nesi, P. (2000). Temporal logics for real-time system specification. *ACM Comput. Surv.*, 32(1), 12-42.
- Bergeson, L. L. (2006). RoHS, WEEE and related EU directives. *Pollution Engineering*, 38(9), 15-15.
- Bernaras, A., Laresgoiti, I., Bartolome, N., & Corera, J. (1996). Ontology for fault diagnosis in electrical networks. *Proceedings of the International Conference on Intelligent Systems Applications to Power Systems*, 199-203.
- Bernstein, P. A. & Goodman, N. (1983). Multiversion concurrency control - theory and algorithms. *ACM Trans. Database Syst.*, 8(4), 465-483.
- Bernstein, P. A., Shipman, D. W., & Wong, W. S. (1979). Formal Aspects of Serializability in Database Concurrency Control. *Software Engineering, IEEE Transactions on*, SE-5(3), 203-216.
- Boothroyd Dewhurst Inc. (2007). DFMA® Design For Manufacture and Assembly. Retrieved 17/09/2007, from <http://www.dfma.com/software/index.html>
- Carey, M. J. (1983). Granularity hierarchies in concurrency control. *Proceedings of the 2nd ACM SIGACT-SIGMOD symposium on Principles of database systems*, 156-165.
- Carey, M. J. & Livny, M. (1989). Parallelism and concurrency control performance in distributed database machines. *Proceedings of the 1989 ACM SIGMOD international conference on Management of data*, 122-133.

- Ceria, S., Nobili, P., & Sassano, A. (1998). A Lagrangian-based heuristic for large-scale set covering problems. *Math. Programming*, 81(2, Ser. B), 215--228.
- Chandrasekaran, B., Josephson, J. R., & Benjamins, V. R. (1999). What are ontologies, and why do we need them? *Intelligent Systems and Their Applications, IEEE [see also IEEE Intelligent Systems]*, 14(1), 20-26.
- Charles, M. (1982). Preemptive Scheduling with Release Times, Deadlines, and Due Times. *J. ACM*, 29(3), 812-829.
- Chen, Y. M. (1997). Development of a computer-aided concurrent net shape product and process development environment. *Robotics and Computer-Integrated Manufacturing*, 13(4), 337-360.
- Chen, Y. M. & Tsao, T. H. (1998). A structured methodology for implementing engineering data management. *Robotics and Computer-Integrated Manufacturing*, 14(4), 275-296.
- Chonoles, M. J. & Quatrani, T. (1996). *Succeeding with the Booch and OMT methods : a practical approach; Lockheed Martin Advanced Concepts Center, Rational Software Corporation*: Addison-Wesley.
- Chou, H. T. & Kim, W. (1986). Unifying framework for version control in a CAD Environment. *Proceeding of Twelfth International Conference on Very Large Data Bases*, 336-344.
- CIMdata. (1996). *Product Data Management: The Definition, An Introduction to Concepts, Benefits, and Terminology* (Fourth ed.): CIMdata.
- CIMdata. (2005). *SolidWorks Office professional: PDMWorks*. SolidWorks Corp.
- Dasgupta, P. & Kedem, Z. M. (1983). *A Non-Two-Phase Locking Protocol for Concurrency Control in General Databases*. Paper presented at the Proceedings of the 9th International Conference on Very Large Data Bases.
- Dassault Systèmes SolidWorks Corporation (2008). eDRAWINGS. from <http://www.edrawingsviewer.com/>
- Date, C. J. (2004). *An introduction to database systems* (8th ed.). Pearson/Addison Wesley.
- Deepali, K., Krishna, G. M., Ulka, S., & Venkatesh, R. (2005). Visual specification and analysis of use cases. *Proceedings of the 2005 ACM symposium on Software visualization*, 77-85.
- Ding, Y. & Foo, S. (2002). Ontology research and development. Part 1 - a review of ontology generation. *Journal of Information Science*, 28(2), 123-136.

- Disterer, G. (2002). Management of project knowledge and experiences. *Journal of Knowledge Management*, 6(5), 512.
- Dixon, C. (2006). Using temporal logics of knowledge for specification and verification--a case study. *Journal of Applied Logic*, 4(1), 50-78.
- Drusinsky, D. (2006). *Modeling and verification using UML statecharts: a working guide to reactive system design, runtime monitoring and execution-based model checking*: Newnes.
- Eden, A. H. & Hirshfeld, Y. (2001). *Principles in formal specification of object oriented design and architecture*. Proceedings of the 2001 conference of the Centre for Advanced Studies on Collaborative research, p.3, November 05-07, 2001, Toronto, Ontario, Canada.
- Eich, M. H. (1988). Graph directed locking. *Software Engineering, IEEE Transactions on*, 14(2), 133-140.
- Enovia MatrixOne (2006). Materials Compliance Central. Retrieved 19/09/2007, from <http://www.matrixone.com/matrixonesolutions/materialscompliancencentral.html>
- Eswaran, K. P., Gray, J. N., Lorie, R. A., & Traiger, I. L. (1976). The notions of consistency and predicate locks in a database system. *Commun. ACM*, 19(11), 624-633.
- Eynard, B., Gallet, T., Nowak, P., & Roucoules, L. (2004). UML based specifications of PDM product structure and workflow. *Computers in Industry*, 55(3), 301-316.
- Farrell, M. W. & Myers, J. R. (1981). Applying structured tools and techniques to the development of software for a small computer system. *Proceedings of the 1981 ACM SIGSMALL symposium on Small systems and SIGMOD workshop on Small database systems*, 1-8.
- Fawzi, H. (2007). Networks as a means of supporting the adoption of organizational innovations in SMEs: the case of Environmental Management Systems (EMSs) based on ISO 14001. *Corporate Social Responsibility and Environmental Management*, 14(3), 167-181.
- Garcia, A. C. B., Kunz, J., Ekstrom, M., & Kiviniemi, A. (2004). Building a project ontology with extreme collaboration and virtual design and construction. *Advanced Engineering Informatics*, 18(2), 71-83.
- Gary, J. N., Lorie, R. A., & Putzolu, G. R. (1975). Granularity of locks in a large shared data base. *Proceedings of the International Conference on Very Large Data Bases*, 428-451.
- Gary, T. L., Tim, W., & Albert, L. B. (1999). Formal semantics for SA style data flow diagram specification languages. *Proceedings of the 1999 ACM symposium on Applied computing*, 526-532.

- Goel, S., Bhargava, B., & Madria, S. K. (2000). An adaptable constrained locking protocol for high data contention environments: correctness and performance. *Information and Software Technology*, 42(9), 599-608.
- Gomaa, H. (1984). A software design method for real-time systems. *Commun. ACM*, 27(9), 938-949.
- Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2), 199-220.
- Guarino, N. (1998). Formal Ontology and Information Systems. *Proceedings of the 1st International Conference on Formal Ontologies in Information Systems, FOIS'98, Trento, Italy. IOS Press*, 3-15.
- Harris, S. B. (1996). Business strategy and the role of engineering product data management: a literature review and summary of the emerging research question. *Journal of Engineering Manufacture*, 210(B3), 207-220.
- Hughes, G. E. & Cresswell, M. J. (1968). *An introduction to modal logic*: Methuen.
- IBM Corporation (2007). ENOVIA SmarTeam. Retrieved 16/09/2007, from <http://www-306.ibm.com/software/applications/plm/smarteam/>
- Jun, W. C. (2000). A multi-granularity locking-based concurrency control in object-oriented database systems. *Journal of Systems and Software*, 54(3), 201-217.
- Kedem, Z. & Silberschatz, A. (1979). Controlling concurrency controlling using locking protocols. *Annual Symposium on Foundations of Computer Science*, 274-285.
- Kedem, Z. & Silberschatz, A. (1980). *Non-two-phase locking protocols with shared and exclusive locks*. Very Large Data Bases Conference, 309-317.
- Keller, G. & Teufel, T. (1998). *SAP R/3 process-oriented implementation: iterative process prototyping/ translated by Audrey Weinland*. Harlow: Addison Wesley Longman.
- Kim, J. A., Kim, J. H., & Park, N. (1998). Development of PDM framework and customization environment. *Technology of Object-Oriented Languages, 1998. TOOLS 28. Proceedings*, 40-49.
- Kim, S. H., Oh, T. H., & Park, J. Y. (1997). The object-oriented modeling for product data management. *Computer Applications in Production and Engineering*, 33-46.
- Kuo, F. Y. & Karimi, J. (1988). User interface design from a real time perspective. *Commun. ACM*, 31(12), 1456-1466.

-
- Langer, A. M. (2008). *Analysis and design of information systems* (3rd ed.). Springer.
- Lee, S. Y. & Liou, R. L. (1996). A multi-granularity locking model for concurrency control in object-oriented database systems. *IEEE Transactions on Knowledge and Data Engineering*, 8(1), 144-156.
- Leistner, M. (1999). The Growth and Environment Scheme. *Greener Management International* (27), 79.
- Leong, K. K. (2002). *An architecture for web-based distributed product data management*. MPhil thesis, Dept. of Industrial & Systems Engineering, The Hong Kong Polytechnic University, Hong Kong.
- Leong, K. K., Yu, K. M., & Lee, W. B. (2003). A security model for distributed product data management system. *Computers in Industry*, 50(2), 179-193.
- Li, O. K. V. (1987). Performance models of timestamp ordering concurrency control algorithms in distributed databases. *IEEE Transactions on Computers*, 36(9), 1041-1051.
- Liou, D. J. E. (1994). *An object-oriented database approach to manufacturing information systems with emphasis on production management*. PhD thesis, Arizona State University.
- Liu, C. L. & James, W. L. (1973). Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *J. ACM*, 20(1), 46-61.
- Manji, J. F. (1995). Data/document management: Making PDM pay. *Machine Design*, 67(11), 81.
- Manna, Z. & Pnueli, A. (1992). *The temporal logic of reactive and concurrent systems*. Springer-Verlag.
- Manna, Z. & Wolper, P. (1984). Synthesis of Communicating Processes from Temporal Logic Specifications. *ACM Trans. Program. Lang. Syst.*, 6(1), 68-93.
- Mansour, Z., Val, C., & Dale, C. (1995). A survey of current object-oriented databases. *SIGMIS Database*, 26(1), 14-29.
- Miller, E., MacKrell, J., & Mendel, A. (1999). *PDM buyer's guide: product data management systems for improving processes and products* (7th ed.). CIMdata.
- Mohan, C., Fussell, D., Kedem, Z. M., & Silberschatz, A. (1985). Lock conversion in non-two-phase locking protocols. *IEEE Transactions on Software Engineering*, SE-11(1), 15-22.

- Ng, S. H. & Hung, S. L. (1995). Multigranularity locking in multiple job classes transaction processing system. *SIGMOD Rec.*, 24(1), 27-32.
- Noy, N. F. & McGuinness, D. L. (2003). Ontology Development 101: A Guide to Creating Your First Ontology. from http://protege.stanford.edu/publications/ontology_development/ontology101.html
- OMG (2007). Object Management Group Unified Modeling Language Specification Version 2.1.1. from <http://www.uml.org/#UML2.0>
- Pahng, F., Senin, N., & Wallace, D. (1998). Distribution modeling and evaluation of product design problems. *Computer-Aided Design*, 30(6), 411-423.
- Partidario, P. J. & Vergragt, J. (2002). Planning of strategic innovation aimed at environmental sustainability: actor-networks, scenario acceptance and backcasting analysis within a polymeric coating chain. *Futures*, 34(9-10), 841-861.
- Patil, L., Dutta, D., & Sriram, R. (2005). Ontology-based exchange of product data semantics. *IEEE Transactions on Automation Science and Engineering*, 2(3), 213-225.
- PE International (2007). The new Generation GaBi 4. Retrieved 27/09/2007, from <http://www.gabi-software.com/gabi/gabi-4/>
- Philip, A. B. & Nathan, G. (1981). Concurrency Control in Distributed Database Systems. *ACM Comput. Surv.*, 13(2), 185-221.
- Philpotts, M. (1996). An introduction to the concepts, benefits and terminology of product data management. *Industrial Management & Data Systems*, 96(4), 11.
- Pnueli, A. (1981). The temporal semantics of concurrent programs. *Theoretical Computer Science*, 13(1), 45-60.
- PRé Consultants (2007, 18/09/2007). SimaPro 7. Retrieved 17/09/2007, from http://www.pre.nl/simapro/simapro_lca_software.htm
- Prior, A. (1967). *Past, Present and Future*. Oxford University Press.
- Reisdorph, K. (1999). Why use version control? . *Delphi Developer's Journal*, 5(1), 14-16.
- Rezayat, M. (2000). The Enterprise-Web portal for life-cycle support. *Computer-Aided Design*, 32(2), 85-96.
- Ryan, D. M. & Foster, B. A. (1981). Integer Programming Approach To Scheduling. *Computer Scheduling of Public Transport, Urban Passenger Vehicle and Crew Scheduling*, 269-280.

-
- Salkin, H. M. (1989). *Foundations of integer programming / Harvey M. Salkin, Kamlesh Mathur ; with contributions by Robert Haas*. North-Holland.
- Sayre, D. (1996). *Inside ISO 14000 : the competitive advantage of environmental management / Don Sayre*. Delray Beach, Fla. :: St. Lucie Press.
- Shaw, H. K., Susan Bloor, M., & Pennington, A. (1989). Product Data Models. *Research in Engineering Design*, 1(1), 43-50.
- Siemens (2008). Explore the Greater Powers of Teamcenter Solutions. from http://www.plm.automation.siemens.com/en_us/products/teamcenter/solutions_by_product/index.shtml
- SolidWorks (2004). *SolidWorks 2005 PDMWorks*.
- SolidWorks (2005). *SolidWorks Office professional: PDMWorks*. SolidWorks Corp.
- Sree, N., Rebecca, W., & Prebhu, G. M. (1990). Knowledge-based graph theoretic analysis of data flow diagrams: integrating CASE tools with expert systems. *Proceedings of the 1990 ACM SIGBDP conference on Trends and directions in expert systems*, 58-71.
- Stadlbauer, H. (1992). Product data model for design support using functional skeletons. *The 1992 ASME International Computers in Engineering Conference and Exposition; San Francisco, CA; USA; 02-06 Aug. 1992*, 149-154.
- Talens, G., Chabane, O., & Colinas, M. F. (1993). Versions of Simple and Composite Objects. *Proceedings of the 19th International Conference on Very Large Data Bases*, 62-72.
- Tao, Y. & Kung, C. (1991). Formal definition and verification of data flow diagrams. *Journal of Systems and Software*, 16(1), 29-36.
- Thomasian, A. (1998). Concurrency control: methods, performance, and analysis. *ACM Comput. Surv.*, 30(1), 70-119.
- UGS Corporation (2007). Compliance Management. from http://www.ugs.com/en_us/products/teamcenter/solutions_by_product/compliance_management.shtml
- Vazquez, F. (1994). Identification of complete data flow diagrams. *SIGSOFT Softw. Eng. Notes*, 19(3), 36-40.
- Wieringa, R. (2003). *Design methods for reactive systems : Yourdon, Statemate, and the UML / R.J Wieringa*. San Francisco, Calif. :: Morgan Kaufmann Publishers.
- Won, K., Bertino, E., & Garza, J. F. (1989). Composite objects revisited. *ACM SIGMOD Record* 18, 337-347.

- Wood, W. G. (1990). Temporal logic case study In *Lecture Notes in Computer Science* (Vol. 407, pp. 257-263): Springer Berlin / Heidelberg.
- Yannakakis, M., Papadimitriou, C. H., & Kung, H. T. (1979). Locking policies: Safety and freedom from deadlock. *Annual Symposium on Foundations of Computer Science* 286-297.
- Yeh, S. C. & You, C. F. (2002). STEP-based data schema for implementing product data management system. *International Journal of Computer Integrated Manufacturing*, 15(1), 1-17.
- Yourdon, E. (1979). *Structured design: fundamentals of a discipline of computer program and systems design*. Prentice Hall.
- Yu, P. S., Wu, K. L., Lin, K. J., & Son, S. H. (1994). On real-time databases: concurrency control and scheduling. *Proceedings of the IEEE*, 82(1), 140-156.
- Zhang, S., Shen, W., & Ghenniwa, H. (2004). A review of Internet-based product information sharing and visualization. *Computers in Industry*, 54(1), 1-15.
- Zhang, X., Wang, T., Wan, L., & Zhou, J. (1995). Open architecture and implementation for product data management. *High Technology Letters*, 1(2), 1-6.
- Zhao, W., Ramamritham, K., & Stankovic, J. A. (1987). Preemptive scheduling under time and resource constraints. *IEEE Transactions on Computers*, C-36(8), 949-960.

Appendix A

eDrawings Plug-in

The types of features and functions of “eDrawings” plug-in being used in developing the environmental compliance system presented in Chapter 4, are described in this appendix. eDrawings [Dassault Systèmes SolidWorks Corporation 2008] is a free e-mail-enabled program included with SolidWorks that let users share designs with the people who need to see to them. It is available in the standard or professional version; the standard version gives the user the power to view, create and share 3D models and 2D drawings in eDrawings (eDRW, ePRT, eASM), DXF, and DWG format files. This version can be downloaded freely from www.eDrawingsViewer.com. The professional version includes additional capabilities to markup and measure a model, user license can be purchased from the website.

Key Features of eDrawings Plug-in

Everyone in the design process can collaborate more effectively with eDrawings and various CAD software programs using the following features.

Creating an eDrawings file

To make a file viewable with eDrawings viewer, the corresponding drawing file must be published using the eDrawings publisher. By saving the file that is already open in a CAD application into the one of the following appropriate formats:

- Part document. Save as **eDrawings (*.eprt)**
- Assembly document. Save as **eDrawings (*.easm)**
- Drawing document. Save as **eDrawings (*.edrw)**

Sending an eDrawings file

eDrawings files can be sent to other using send function in eDrawing viewer. The email contains detailed instructions for the recipient about how to use the eDrawings application. By sending the files in an email using one of the following options, the receiver does not need to download anything to view the files.

- **eDrawings file (.edrw, .eprt, .easm).** Attaches a copy of the active document saved as an eDrawings file.
- **Zip (.zip).** Attaches a copy of the file as an executable (.exe) file, saved in a zip file. The recipient needs an application to unzip the file.
- **HTML page (.htm).** Attaches a copy of the file as an HTML file. If your default outgoing mail format is set to HTML, the eDrawings HTML is embedded directly in the email.
- **Executable (.exe).** Attaches a copy of the file as an executable (.exe) file.

Markup Tools

Markup tools is an optional tool that enhances the eDrawings application and is only available in professional version. Notes, dimension (Figure A.1), text (Figure A.2), and graphical elements to the model can be added using the markup tools. eDrawings automatically saves markup files as threaded comments with the eDrawings file. Markup data can be saved separately, as a markup file, without the models. In such a way, it facilitates the reviewing process when several people review the file. For example, reviewers can add comments, and then send you only a markup file.

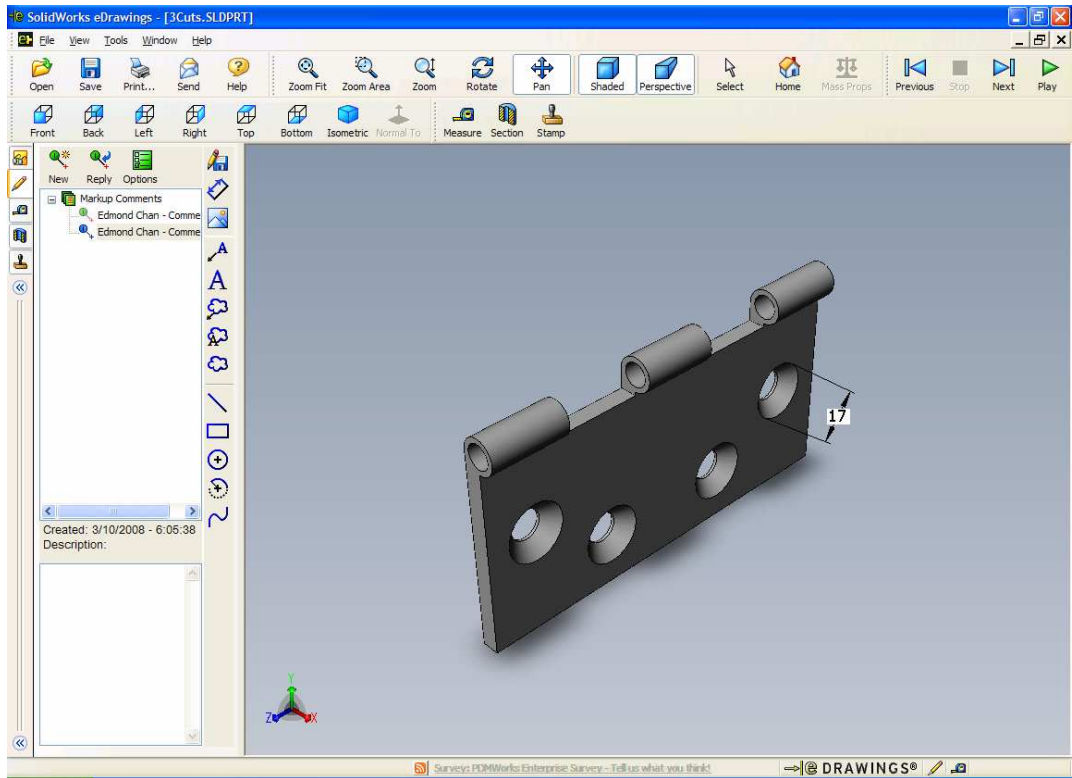


Figure A.1 Showing dimension of a hole using Dimension function

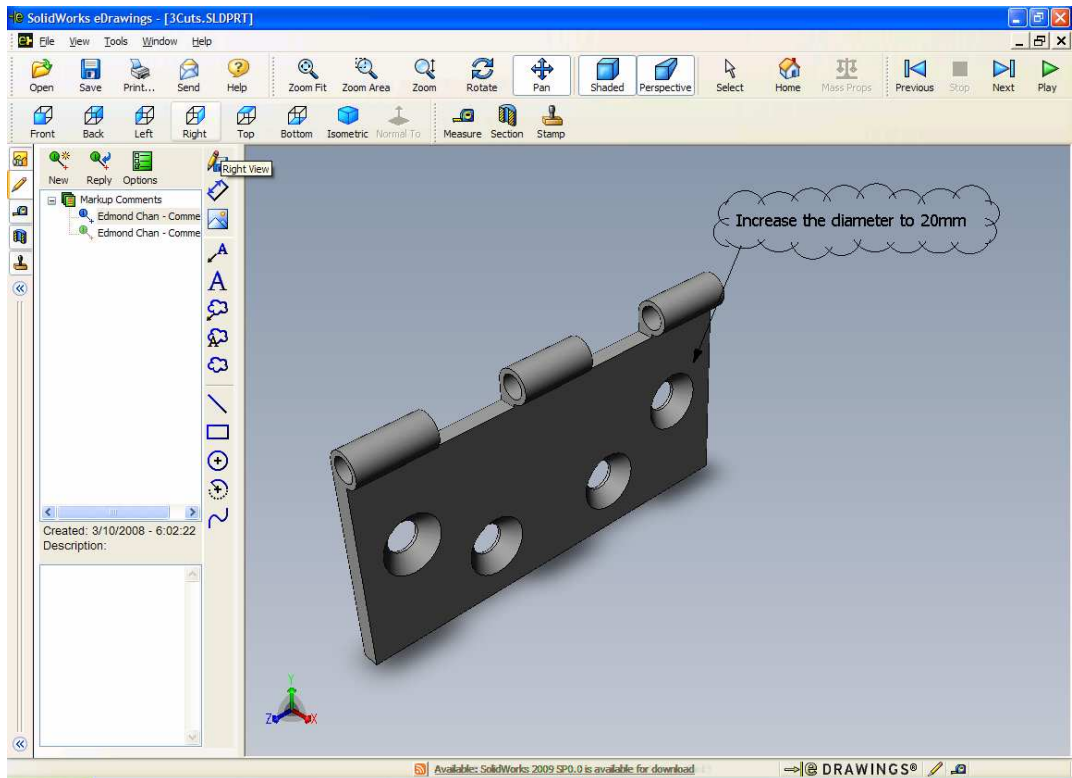


Figure A.2 Adding comment to a document

Appendix B

Constraint Branching

In Chapter 8, given an optimal fractional solution to the linear program relaxation, an integer solution may be found using branch and bound. Effective branching strategies have been developed to solve complex problems in a reasonable amount of time.

The conventional variable branching strategy [Salkin 1989] can be used to solve the fractional problem. Each node of the branch and bound tree corresponds to a linear programming problem in which certain variables have been chosen and constrained to take integer values. In the branching process, a selected node is branched on by imposing two constraints separately on a single fractional variable x_f , these constraints are

$$\begin{aligned} x_f &\leq \lfloor x_f \rfloor && (0\text{-branch}) \\ x_f &\geq \lceil x_f \rceil && (1\text{-branch}) \end{aligned}$$

and two new linear programming problems are created as the result.

For set partitioning problems, the 0-branch constraint has minimal effect on the objective function, as there are likely to be many alternative variables available to enter the basis at little cost. Because the objective value hardly increases on the 0-branch, the bounding process will be less effective. For large-scale problem, it is likely that there will be many nodes and branches explored to find an integer solution. Contrarily, the 1-branch effectively forces the variable to the value of one. This has a significant effect on the solution to the 1-branch problem and often increases the objective function, which can aid the bounding process. Variables that cover any constraint covered by x_f will be eliminated and fewer numbers of variables need to be priced as the branching process goes on.

An alternative approach is to use the constraint branching method developed by Ryan and Foster [Ryan & Foster 1981]. The basic concept of constraint branching is to branch on a set of variables rather than a single variable, the set is defined by a pair of constraints (s and t). The branch is defined by

Identify a pair of constraints s and t

$$s.t. \quad 0 < \sum_{\substack{j \in J(s,t) \\ j \in \text{Basis}}} x_j < 1$$

$$\text{where } J(s,t) = \{j \mid a_{sj} = 1 \text{ and } a_{tj} = 1\}$$

The 0-branch indicates that s and t must not be covered by a single variable and is implemented by banning all variables in $J(s,t)$. Alternatively, the 1-branch indicates that s and t must be covered together and is implemented by forcing variables in the complementary set $\bar{J}(s,t)$ to be zero, where

$$\bar{J}(s,t) = \{j \mid (a_{sj} = 1 \text{ and } a_{tj} = 0) \text{ or } (a_{sj} = 0 \text{ and } a_{tj} = 1)\}$$

A standard method to determine the branch in set partitioning problem is to select s and t , so that both s and t are equality constraints and the objective function value of the linear programming problem is maximized. Depth-first 1-branch search is implemented to evaluate the ranch and bound tree.

The constraint branching strategy is more effective than the conventional variable branching. Many variables are eliminated from the problem on both the 0-branch and 1-branch, since the set of variables are banned on either branch, this allows a balanced tree to be developed and the bounding process on 0-branch becomes as effective as on 1-branch. This branching strategy requires fewer branches than the conventional strategy. The constraint branching strategy has been found to be effective in solving large-scale integer programming problems

Bounding Procedure and Termination Criterion

The final aspect of the branch and bound process used for the problem is to determine the bounding procedure and tree search termination criteria. The bounding procedure applied is to use the objective value of the integer solution found as a bound during the tree search. Beside the bounding procedure, there are two related tree search termination criteria. If an integer solution is found excesses within some percentage of the relaxed linear programming solution, then the branch and bound process is terminated. Another criterion is the number of nodes searched in the process. If a number of nodes searched reach the predefined limit, then the process is terminated. Such premature terminations are to recognize that the most important point of the branch and bound process is to find a feasible integer solution in a reasonable amount of time. Although better solutions may exist elsewhere in the branch and bound tree, the extra time required to find such solution can be substantial and result in little improvement.