

DEVELOPMENT OF A NETWORK-INTEGRATED FEATURE-DRIVEN ENGINEERING ENVIRONMENT

WANG GUOXIAN

NATIONAL UNIVERSITY OF SINGAPORE

2006

**DEVELOPMENT OF A NETWORK-INTEGRATED
FEATURE-DRIVEN ENGINEERING ENVIRONMENT**

WANG GUOXIAN
(M.E. Tsinghua University)

**A THESIS SUBMITTED
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
DEPARTMENT OF MECHANICAL ENGINEERING
NATIONAL UNIVERSITY OF SINGAPORE**

2006

SUMMARY

The contemporary product design-to-manufacturing process involves a group of knowledge-intensive applications and functions. A distributed concurrent and collaborative engineering environment is thus desirable to assist the integration of all the phases of engineering activities together. System integration via network communications has been intensively studied. However, the challenges are still tremendous and the solutions vary in different application contexts and different development practices performed by different researchers. There are very few formulated system patterns to follow or effective approaches to dictate addressing relevant issues with good traceability from functional requirements to system implementation details.

This thesis presents an effort to develop a network-integrated engineering environment while emphasizing on the pursuit of a formulated system integration approach with promising applications to a broad range of engineering process types. Collectively, this range of processes is called feature-driven engineering processes, every sub-process within which involves the handling of feature-based models, either feature model creation, feature model mapping, or model transformation from feature-based models to ordinary geometrical models. The proposed integration approach is centered on a concept of CAX framework which borrows ideas from the CAD framework, a notion widely used in the area of EDA (Electronic Design Automation) to turn collections of individual electronic design tools into coherent, effective and user-friendly design environments. The study was conducted in the context of developing a prototype for

CAD/CAM of progressive dies. It has been treated as the vehicle for validating the key concepts proposed in this research.

Development of the desired integrated engineering environment based on the CAX framework approach began from characterizing the feature-driven engineering processes. This includes process decomposition, analysis, modeling and re-engineering, and identification of special properties required to be taken into account. The characterization effort in this study generates a group of IDEF0 activity models, a set of design change propagation properties and a special design transaction model. The key for complete system specification is to conceptually construct the CAX framework, which provides interfaces for all participating engineering tools. The framework consists of a workbench application accessible by all tool users, the framework kernel, a management database, and the raw design data base. Two steps are taken for framework construction. The first step is to make all implementation decisions to conceptualize a “skeletal” framework with the management database schema being empty. The second step is to develop the management database schema or relevant information models and further make the database coherently co-work with other components in the framework. Object-orientation has permeated the full system development process from beginning to end.

The information models for database schema include two parts: one for realizing PDM (Product Data Management), the other for process management. The full course of information modeling was incremental, *i.e.*, PDM, process management, and overall. The kernel of the PDM model is a novel design versioning scheme supporting design change propagation management. For the process management, it is modeled as a

semi-structured design flow allowing dynamic specification while the process is in execution. For the examination of the integration capabilities of the derived network-integrated engineering environments, especially on how CE (Concurrent Engineering) strategy is supported, a demonstration session running on the developed prototype was worked out. The results show that the system exhibits advantages, which indirectly demonstrates the effectiveness of the proposed CAX framework integration approach.

The thesis is concluded by a recommendation for CAD/CAM system developers to adaptively use this approach in other comparable areas if their targeted design-to-manufacturing process can be roughly classified as a feature-driven engineering process.

ACKNOWLEDGEMENTS

First of all, I would like to express my gratitude to my supervisor, Professor Andrew Y. C. Nee, for his encouragement, guidance, support and valuable advice during the whole course of my research. It has been a learning experience for me working with him, not only academically, but also in other aspects of life such as career development. I would also like to thank Dr. Zhang, Wenzu for his enthusiasm, support and assistance for this research.

Special thanks are due to Dr. Cheok, Beng Teck and Dr. Lu, Chun for their supportive co-supervision to my graduate study which was partly undergone at the Institute of High Performance of Computing.

Thanks to my wife and my son for their suffering from my long absent spells in their family life over the years.

Finally, I wish to thank the National University of Singapore and the Institute of High Performance Computing for giving me the opportunity to pursue this Ph.D. degree with financial support.

TABLE OF CONTENTS

Summary	i
Acknowledgements	iv
Table of Contents	v
List of Figures	x
List of Tables	xiii
List of Acronyms	xiv
List of Notations	xvi
CHAPTER 1 INTRODUCTION	1
1.1. “Integrated View” of a Computer-integrated Engineering Environment	2
1.1.1. Evolvement of the CAPDE	3
1.1.2. The Roles of Feature Modeling and Mapping Technologies in CAPDE	5
1.1.3. The Need for an Advanced Integration Infrastructure and Associated System Building-up Methodologies	7
1.2. Research Objectives, Expected Outcomes and Research Scope	9
1.2.1. Summary of the Open Issues for Integrating Feature-driven Engineering Processes in Terms of Published Literature	10
1.2.2. Research Problem Statement	11
1.2.3. Development of a Prototype with Long-term Objectives for Industry Applications	12
1.2.4. Theoretical Values of the Present Research	14
1.2.5. Other Potential Application Areas of the Research	15
1.2.6. Research Scope and Overall Approach	16
1.3. Terminology Statement	18
1.4. Thesis Organization	21

CHAPTER 2 LITERATURE REVIEW	22
2.1. A Historical Perspective on System Integration from Design to Manufacturing	22
2.2. Some Aspects Driving System Integration from Design to Manufacturing.	24
2.3. Review of Several Representative Integration Architectures	45
CHAPTER 3 CHARACTERIZING FEATURE-DRIVEN ENGINEERING PROCESS.....	55
3.1. Hacking the Complex Engineering Process: the Feature-driven Way.....	55
3.2. Process Decomposition and Information Flow	61
3.2.1. Moving Some Design Tasks in One Sub-Process ahead to Enter Its Upstream Sub-Process	62
3.2.2. Formulated Process Decomposition and Information Flow: a Comprehensive IDEF0 Activity Model.....	67
3.3. Interdependence Semantics and Design Change Propagation Property.....	74
3.3.1. Global View of Interdependence Semantics in a Feature-driven Process: Design Object Derivation Graph	74
3.3.2. Expanding the Feature Transformation Taxonomy Towards Dependency Relationship Taxonomy.....	78
3.3.3. Model Derivation Function	79
3.3.4. Design Change Propagation Property	82
3.4. A Special Design Transaction Model for Feature-driven Engineering Process	84
3.4.1. The Means by Which an Engineering Tool Manipulates Relevant Data through Design Sessions	85
3.4.2. Basic Design Transaction Model.....	87

3.4.3. A Special Design Transaction Model for Feature-driven Engineering Process.	88
3.4.4. Discussions on the Proposed Design Transaction Model.....	91
CHAPTER 4 OVERVIEW OF THE CAX FRAMEWORK INTEGRATION	
APPROACH	93
4.1. Rationale of the CAX Framework Approach.....	94
4.2. Definition of Functional Requirements and System Architecture.....	97
4.2.1. Functional Requirements.....	97
4.2.2. Some Basic Strategies for Defining the General Framework Architecture	98
4.2.3. The General System Architecture	101
4.3. A Roadmap of Implementation and the “Skeletal” Framework	102
4.3.1. A Roadmap of Implementation	102
4.3.2. Functionality Partition between the Client and the Server.....	103
4.4. Some Basic Implementation Decisions for the CAX Framework-based	
Network-integrated Engineering Environment.....	105
4.4.1. Platform and Programming Language.....	106
4.4.2. The Wrapper and the Way to Make the CAX Tools Available on the Internet	107
4.4.3. DBMS for the Management Database.....	110
4.4.4. File Transfer	111
CHAPTER 5 VERSION CONTROL AND CONFIGURATION	
MANAGEMENT	113
5.1. Version Control and Configuration Management Concepts.....	113
5.2. A Version Control and Configuration Management Model.....	116
5.2.1. Basic Concepts	116
5.2.2. Design Change Propagation Scope and Object Version Identification.....	119
5.2.3. Control of Configuration Version Creation.....	125

5.3. Specification of Operations	127
5.3.1. Operations on Projects.....	128
5.3.2. Operations on Configurations.....	129
5.3.3. Operations on Design Objects	130
5.4. Application of the Proposed Model in the Integrated Progressive Die Design and Manufacturing Engineering Environment	132
5.5. Towards a Comprehensive Information Model and a Full-fledged GUI Design.....	139
CHAPTER 6 ENGINEERING PROCESS MANAGEMENT	140
6.1. A Process Management Mechanism Based on Design Flow Configuration	140
6.1.1. Overview	141
6.1.2. Process Representation.....	145
6.1.3. The Process Execution Engine	148
6.2. A Comprehensive Information Model.....	150
6.3. Two UML Sequence Diagrams Highlighting the Basic Process Management Functionality	157
CHAPTER 7 WORKBENCH GUI DESIGN AND SOME EXPERIMENTAL RESULTS	164
7.1. The Scope of the Demonstration Session.....	164
7.2. Description of the Results for the Principal Demonstration Steps	167
7.3. Discussions.....	178
7.3.1. Evaluation.....	178
7.3.2. Concurrent Engineering Support.....	179
7.3.3. Further Predictable Capabilities	180

CHAPTER 8 CONCLUSIONS.....	183
8.1 Research Contributions and Discussions	183
8.2 Limitations	188
8.3 Future Directions.....	190
References.....	193
Publications from This Research.....	211

LIST OF FIGURES

Fig. 1.1 Evolvment of the computer-assisted product development environment (Tian <i>et al.</i> , 2002)	5
Fig. 2.1 Systematic procedure to develop a software system using the OO paradigm.	43
Fig. 2.2 (a) Modular integration of CAD/CAM software (MICS)	46
(b) The Pro/E-ToolPro CAD/CAM system (PTCS) (Thomas & Fischer 1996)	46
Fig. 2.3 The SDM and IPDE architecture (Urban <i>et al.</i> , 1996, 199a)	47
Fig. 2.4 An integrated framework for net shape product and process development (Chen 1997)	48
Fig. 2.5 The CONCERT environment (Hanneghan <i>et al.</i> , 1995,1998)	49
Fig. 2.6 Architecture of the SUKITS integrated development environment (Westfechtel 2000)	50
Fig. 2.7 The WWW-based integrated product development platform for sheet metal concurrent design and manufacturing (Xie <i>et al.</i> , 2001)	52
Fig. 2.8 Structure of feature-based collaborative development system (Wang & Zhang 2002)	53
Fig. 3.1 The feature-driven progressive die design and manufacturing process	58
Fig. 3.2 (a) Process decomposition under normal circumstance (Nee & Cheok 2001)	64
(b) Process decomposition following the conventional logic.....	65
(c) Process decomposition after process re-engineering	65
Fig. 3.3 Diagram A-0	68
Fig. 3.4 Diagram A0	68
Fig. 3.5 Diagram A1	71
Fig. 3.6 Diagram A2	72
Fig. 3.7 The Design Object Derivation Graph.....	76
Fig. 3.8 Two types of design changes propagation.....	77
Fig. 3.9 Four possible means by which a tool manipulates relevant data.....	86
Fig. 3.10 The basic design transaction model (Wolf 1994).....	88

Fig. 4.1 The general system architecture (Wolf 1994)	101
Fig. 4.2 Creation of client/server with Java RMI.....	104
Fig. 4.3 Integrating a tool with the framework kernel through a wrapper.....	108
Fig. 5.1 Product configurations.....	114
Fig. 5.2 Multi-version database as a set of database versions	117
Fig. 5.3 Augmented layered transaction schema for handling engineering data (Developed based on Fig. 6.6 in Wolf (1994))	118
Fig. 5.4 Comparison of two versioning approaches	121
Fig. 5.5 Design change propagation scope and object version identification.....	123
Fig. 5.6 Information structures and the <i>IsProactive</i> attribute	126
Fig. 5.7 The computation logic to control creation of configuration version.....	127
Fig. 5.8 The configuration VDG for the example scenario	132
Fig. 5.9 Step 1 in the scenario: generating <i>Con₂</i> launched by a design change on <i>Product-Feature-Model</i>	134
Fig. 5.10 Step 2 in the scenario: generating <i>Con₃</i> launched by a design change on <i>Die- Operation-Feature-Model</i>	136
Fig. 5.11 Step 3 in the scenario: generating <i>Con₄</i> launched by a design change on <i>Part₄-Feature-Model</i>	138
Fig. 6.1 A meta process model for feature-driven engineering process	146
Fig. 6.2 Example compound design flow containing two activities and a sub-flow ..	147
Fig. 6.3 A comprehensive information model for the example implementation	152
Fig. 6.4 The design flow before (a) and after (b) the CAPP sub-flow is defined	156
Fig. 6.5 A UML sequence diagram highlighting process management functionality (simple design transaction case).....	159
Fig. 6.6 A UML sequence diagram highlighting process management functionality (complex design transaction case).....	161
Fig. 7.1 The snapshot of the user authentication window.....	168
Fig. 7.2 The snapshot of the authentication failing alert window.....	168
Fig. 7.3 Selection of on-line or off-line work-mode.....	168
Fig. 7.4 Locate a project	169

Fig. 7.5 Locate a configuration version	170
Fig. 7.6 Open a configuration version to view its running design flow and composition hierarchy	171
Fig. 7.7 Composition of the overall working window	173
Fig. 7.8 The grouped check-in alert dialog	175
Fig. 7.9 Design state change caused by a grouped check-in	175
Fig. 7.10 A design state at which two activities are concurrently performed.....	176
Fig. 7.11 Creation of a new configuration version	177
Fig. 7.12 The newly created configuration version	178

LIST OF TABLES

Table 5.1 Operations on projects, configurations and design objects..... 128

Table 7.1 Evolvement of a configuration version..... 174

LIST OF ACRONYMS

API	Application Programming Interface;
BOM	Bill Of Materials;
CAD	Computer-Aided Design;
CAM	Computer-Aided Manufacturing;
CAPDE	Computer-Assisted Product Development Environment;
CAPP	Computer-Aided Process Planning;
CAX	Computer-Aided anything;
CBR	Case-Based Reasoning;
CE	Concurrent Engineering;
CFI	CAD Framework Initiative;
CG	Composition Graph;
CIM	Computer-Integrated Manufacturing;
CIFS	Common Internet File System;
CONCERT	CONCurrent Engineering Support;
CSCW	Computer-Supported Cooperative Workspace;
CORBA	Common Object Request Broker Architecture;
DAI	Data Access Interface;
DBV	Database Version;
DCOM	Distributed Common Object Mode;
DFA	Design For Assembly;
DFM	Design For Manufacturing;
DODG	Design Object Derivation Graph;
DPM	Data and Process Management;
EDA	Electronic Design Automation;
ER	Entity Relationship;
GUI	Graphic User Interface;
IDEF	Integration DEFinition;
IPD	Intelligent Progressive Die;
IPDB	Integrated Product Database;
IPPD	Integrated Product and Process Design;
IPDE	Integrated Product Design Environment;
IT	Information Technology;
JVM	Java Virtual Machine;

MEMS	Microelectromechanical System;
MAS	Multi-Agent System;
OO	Object-Oriented or Object Orientation;
OODBMS	Object-Oriented Database Management System;
PC	Personal Computer;
PCB	Printed Circuit Board;
PDM	Product Data Management;
SDM	Shared Design Manager;
STEP	Standard for Exchange of Product model data;
UML	Unified Modeling Language;
UOF	Unit Of Function;
VRML	Virtual Reality Modeling Language;
VDG	Version Derivation Graph;
VM	Virtual Machine;
WM	Workflow Management.

LIST OF NOTATIONS

ΔAB	The information elements in model A with no correspondence in model B ;
ΔBA	The information elements in model B with no correspondence in model A ;
$f(), h()$	Function;
$f_a()$	Adjoint transformation function;
$f_b()$	Conjugate transformation function;
M_A	Model A ;
M_B	Model B ;
M_C	Model C ;
$U(n)$	Design operator through the user interface at time n ;
$X(n)$	The global design state at time n ;
$Y(n)$	The external appearance of the design state in the user interface at time n .

CHAPTER 1

INTRODUCTION

Successful product design and development practice is reflected by the achievement of good design specifications in the design and manufacturing documents (or electronic files) and as short a lead time as possible for the development process. Typically, developing a quality product in a reduced lead time is heavily dependent on the team members' knowledge, the cooperation among them and the tools they use. Among these three factors, the importance of the engineering tools for a company is becoming more outstanding with the constant increase of their functionalities enabled by new information technologies. One trend which can be seen in the past years is that much of the engineers' knowledge has been coded into the computer system and many engineering tasks can be automatically completed by the newly created or revamped intelligent tools. Moreover, many cooperation activities have also become an inner function of the computer-based tools which support strategies, such as CSCW (Computer-Supported Cooperative Workspace). With more task-specific tools being introduced to assist engineering processes, engineers are no longer expected to separately use individual tools. Instead, they are immersed in an integrated engineering

environment consisting of a set of logically related tools, which operate in a coordinated manner.

This thesis presents a systematic approach for the development of network-integrated engineering environments. Due to their complexity, such environments cannot be implemented in an *ad hoc* manner. Rather, their system architectures have to be designed either by following well-formulated patterns or based on creative use of the generic configuration principles of computer-based systems. Formal models have to be built to describe the data and operations of the system both precisely and at a high level of abstraction. Implementation strategies have to be devised to bring together the concepts and technologies involved.

The current introductory chapter is an overview of the thesis. Section 1.1 takes a closer look into the nature of computer-integrated engineering environments, focusing on engineering process decomposition via feature-based modeling and mapping, as well as sub-processes reunification via advanced integration infrastructure. Section 1.2 describes the objective of the research, its expected values and the research scope. Section 1.3 introduces several main fundamental notions used throughout this thesis. Finally, section 1.4 presents an overview of the rest of this thesis.

1.1. “Integrated View” of Computer-Integrated Engineering Environment

Contemporary network-integrated engineering environment has evolved from Computer-Aided Design (CAD) and Computer-Aided Manufacturing (CAM) systems which emerged in the early 1960’s and were originally designed for single users working in isolation to carry out a specific engineering task. One of the most important

underlying thrusts for the evolvement comes from industry's ever-increasing requirements of design automation. While there have been considerable efforts devoted to improve design automation of a complex engineering process by decomposing it into small sub-processes to be easily automated, one can also observe a large number of later yet almost parallel efforts to integrate all the related data, sub-processes, activities, tools and resources so as to automate the process as a whole. Feature-based modeling and mapping plays an important role in engineering process decomposition as well as integration due to its ability to bridge the link between design and manufacturing. Advanced integration infrastructure makes it possible to coordinate and harmonize the activities which go on in the integrated system. Discussed in the following sub-sections are some details about these three interrelated subjects, evolvement of the Computer Assisted Product Development Environment (CAPDE), the roles of feature-based modeling and mapping in CAPDE and the need for advanced integration infrastructure.

1.1.1. Evolvement of the CAPDE

As shown in Fig. 1.1, since the 1970's, there has been a growing trend in manufacturing firms towards the use of computer systems to perform many of the functions related to product design and development. Many types of computer-based engineering tools have been introduced to provide diverse services to the user, with notions, such as CAD, CAPP (Computer-Aided Process Planning), CAM, *etc.* Due to the limitation of information technology in the early days, traditional computer-based engineering tools dominated in providing interactive assistance to a single user to create, modify, store, and render product drawings, virtual solid models or specification documents within a specific engineering discipline in a specific product

life-cycle phase. With the advances of information technology, intelligent abilities were gradually encapsulated into the computer-based tools and the scope of design automation tools was extended from specific applications to integrated systems across disciplines and life-cycle phases (Teti & Kumara 1997). The prevalence of networked computing platforms since the 1990's made another big improvement in that the engineering tools were able to benefit from the distributed computation paradigm. Not only was the engineering environment able to be designed as a monolithic application located on a standalone computer for single users' use, but it was also able to leverage the resources on other computers and/or share information and knowledge with others in a multi-user environment (Regli 1997). Along a parallel trend, the past years also observed the improvements in the understanding of engineering activities from the perspective of application of computer technology. This helps to work out the best way to partition an entire product development process into sub-processes supported by individual tools and then deploy them enterprise- or virtual enterprise-wide so that an optimal integrated engineering environment is finally realized. For example, the Concurrent Engineering (CE) strategy has been used to fine-tune an integrated system by ensuring that the maximum engineering concurrency would be allowed (Prasad 1996). While there are many approaches to use these strategies combining special computing technologies to develop optimal engineering environment, a methodology centered on feature modeling and mapping is especially significant for a wide category of products. The roles of feature modeling and mapping are discussed in the next subsection.

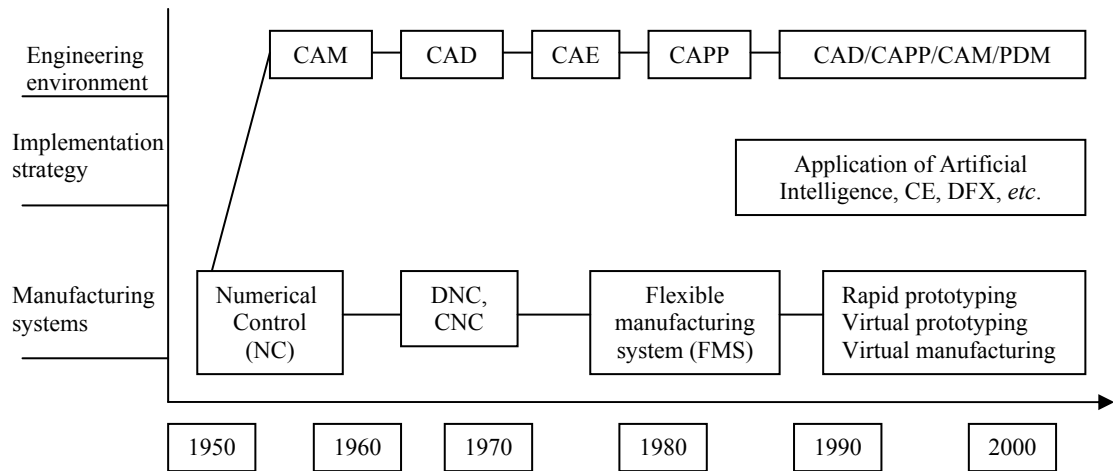


Fig. 1.1. Evolvement of the computer-assisted product development environment
(Tian *et al.*, 2002)

1.1.2. The Roles of Feature Modeling and Mapping Technologies in CAPDE

It is widely recognized that an important point of a product development cycle is to generate an appropriate product information model, which is a common communication medium for designers, analysts, manufactures, and other product development people. The downstream product development data, such as that for tooling, manufacturing, assembly planning, *etc.*, are then generated directly or indirectly from this product model. As such, the information encapsulated in the product model needs to be packageable and transportable among the participating agents in such a way that the intents and concerns of each are neither lost nor unaddressed. Features are seen by many researchers as the natural and most appropriate packaging of design information for manufacturing purposes to bridge the missing link between design and manufacturing (Dixon *et al.*, 1989; Shah 1988). Using features, users can express easily the design intent by manipulating features directly, eliminating tedious intermediate steps. Also, the feature databases allow reasoning systems to perform tasks such as heuristic optimizing, manufacturability analysis, *etc.* It also contains knowledge to facilitate numerical control machine programming,

process planning, and automatic finite element meshing (Shah 1988). In summary, features are an essential component of any intelligent design system (Dixon *et al.*, 1989). According to Hsiao (1990), three methods are used for creating feature-based models to virtually represent a product in CAPDE, namely, human-assisted feature definition, automatic feature recognition/extraction and design-by-features.

The most important significance of feature technology is probably its assistance in engineering process automation. It is generally known that product design and development requires considerable human experience and decision making. Moreover, the engineering activities involved are classified into two types: creative and routine. While the conceptual design process can be seen as creative and too difficult to automate due to a lack of understanding of its nature, the downstream engineering processes are not exactly creative. As a routine design, the sequence of processes is well-structured, and thus feasible to be simulated in an intelligent CAPDE. This strategy is especially useful for a product that has a large portion of its lifecycle in developing its manufacturing process compared to developing its conceptual product model. In another words, it has a long development cycle that can be viewed as a step-wise process chain. Each component process is used to accomplish a part of the engineering tasks, assisted by a specific application which has its own dedicated internal data model and can provide a set of desired engineering renderings. Specialized technology knowledge and *modus operandi* have to be used for problem solving in each component process (Zimmermann *et al.*, 2002). One of the most important types of knowledge is how current tasks are dependent on those carried out by its previous processes or reflected in the data flows, to what extent and in what way the current process data model is dependent on that of the previous processes. Feature-

based model is thus also the best option to be adopted as the corresponding intermediate models for all constituent component processes. This is because it can promote maximum extent of automation when generating these models using an approach called feature mapping (also called feature conversion or feature transformation): generating the new set of feature instances B from the given set A through knowledge-based reasoning supported by feature mapping knowledge base (Zimmermann *et al.*, 2002).

1.1.3. The Need For Advanced Integration Infrastructure and Associated System Building-up Methodologies

Although feature technologies provide a mechanism to bridge the missing link between design and manufacturing, make the consecutive models interoperable and thus allow for expanded design automation across engineering processes, the overall process automation does not come free. Much further effort has to be invested to integrate the constituent feature-based models and the engineering processes for a specific subset of a product development cycle. It can be imagined that in such an engineering environment, engineers work on and manipulate various kinds of feature-based models which have to compatibly work together. More precisely, changes made in one model should be propagated to other models, and an overall integrity for the models must be maintained (Karsai & Gray 2000). Consider the simplest case of an engineering process that is composed of two models a and b and assisted with Tool A and Tool B respectively. Model b is dependent on Model a . Tool A is of feature-based modeling and Tool B is of feature-based mapping. Here, the feature-based modeling and mapping makes the semantic relationship between models a and b understandable by the computer system. Accordingly, much of the design effort can be saved by using

Tool B because of its ability to automatically derive the instance features in b according to its relationship with certain instance features in a. However, there are still several unaddressed factors which affect the design efficiency and productivity. For example, if the two tools are isolated and standalone, it will leave the designer with the problem of frequently entering and exiting two different environments to handle the tools separately since the engineering process is inherently iterative, as well as moving his design data from one tool to the other through file transfer. Furthermore, since there may exist several versions of models a and b in an engineering practice, the engineers should take the responsibility to ensure which pair of a and b are of compatible version throughout the development cycle even after a long period of interval for some reasons. Typically, the real-world product development process is a complex one and there are more than two tools involved. The amount of design data to be handled then multiplies accordingly. Moreover, what is also lacking in the complex real-world case includes the overall support for managing the design process. As a result, the need for complex data and process management in engineering tools integration suggests a need for advanced integration infrastructure.

This need can be explained in that there exist some common functions that have to be shared by the constituent tools in the engineering process. These functions, the product data and process management in the above case, should hold semantics related to the global view of the overall engineering process. Further examination will indicate that the shared functions are unnecessarily limited to these two types aforementioned. A possible alternative is a common knowledge repository function (Zha *et al.*, 2003), the design of which can be considered simultaneously with that of the data management system. Another more general alternative comes from the research on distributed and

collaborative CAX systems, which partition the functions of an application between the client side and the server side. It is found that a large group of distributed engineering applications (tools) usually constitute common modules, such as a solid modeler (Mervyn *et al.*, 2003) which can be deployed and shared on the server side so that the computation efficiency and reusability of generic components can be enhanced. As a result, in order to be compatible with the concept of distributed design (Maropoulos 1995) and provide shared functions for the constituent tools, today's engineering environment is increasingly demanding advanced integration infrastructure. Many other parts should also be integrated into this infrastructure apart from the above core functions, *e.g.*, platforms (computers plus operating system software), physical networks and networking hardware, network protocols and network operating systems. Corresponding to this constantly advancing integration infrastructure, the associated methodologies are also required to be developed to solve all the relevant problems in the course to contrive a soundly integrated system.

1.2 Research Objectives, Expected Outcomes and Research Scope

The need for advanced integration infrastructure and associated system building-up methodologies has prompted remarkable efforts to be devoted to this direction. Chapter 2 gives a comprehensive literature review on these efforts. Observations based on the literature review are summarized and documented in advance at this instant to justify the research objectives, which are presented subsequently. Expected outcomes and research scope are also detailed in this section.

1.2.1. Summary of the Open Issues for Integrating Feature-driven Engineering Processes in Terms of Published Literatures

Although network-integrated product development environment is not a completely new topic in manufacturing engineering, there are still many significant aspects of such an environment that have not been receiving sufficient study. The community still lacks an effective systematic methodology for developing an ideal integrated system to cover the entire product development cycle for a specific type of products, especially one that has a structured feature-driven engineering process. In summary, the following issues are still open.

- There hardly exist any generic and theoretically-strong approaches, following which the system developer can successfully develop a workable system. Most of the systems are given as they are, with no explanation on why these systems are devised in a particular way.
- The functionality of the existing systems does not seem to be comprehensive enough, which is probably due to the fact that the underlying integration infrastructure may not be well-structured and flexible enough. Incorporating further preferable functional modules into an existing system may either be extremely difficult for the system developer because of the overwhelming re-engineering efforts needed, or unwelcome to the end-users because of the unbearable operation complexity.
- Most of the existing systems do not consider the encapsulation of the product development process knowledge, by which the users are able to identify what part of the tasks have been completed, what are ongoing and what are to be done next. Management of the process is fully up to the end-user, who may lose control in the complex and iterative product development process.
- Data integration in most existing systems does not seem to operate at a well-defined granularity level. It either operates at too fine a granularity level (such as

- dots, lines, *etc.*), which makes the system inflexible, or at too coarse a granularity level (such as isolated electronic documents), which makes the system too loosely integrated such that heavy external coupling is required (Liang *et al.*, 1999). It can also be noticed that much attention has been given to avoid traditional piecemeal implementation which causes the engineering environments to become a group of “automation islands”, but very few works have dealt with another important issue of avoiding hard-binding resilient modules together into a rigid monolithic super-tool.
- Most of the systems do not make full use of existing and newly-emerging information technologies, such as the OO (Object-Oriented or Object Orientation) modeling technologies, knowledge-based techniques and the Internet-based technologies. The product database management was either not taken into account or too limiting to provide strong knowledge reuse functions based on rich representation schemata and/or sufficient inference facilities. The performance of the system also needs to be further improved to meet the end-user’s ever-demanding requirements.

1.2.2. Research Problem Statement

The objective of the research reported in this thesis was to study the integrated product development environment in the context of using a new approach that has a strong theoretical foundation. This approach is borrowed from the field of Electronic Design Automation (EDA). The key notion for this approach in its original area is related to using a CAD framework to integrate diverse logically related electronic CAD tools. In design automation in manufacturing engineering, not only CAD tools are involved, but other types of tools, such as that of CAM, CAE, CAPP, *etc.*, may also be involved. The

collection of CAD, CAM, CAE and many other tools is usually called CAX, which means “computer-aided anything”. The notion of the CAD framework is extended to the CAX framework in the current study. Specifically, the problems that are mainly investigated in this research include:

- How can a CAD framework methodology be conceptually applicable to the development of an integrated engineering environment for products which have a feature-driven process?
- What are the adaptations that should be made to tailor the CAD framework to the CAX framework?
- How to use the CAX framework to develop a network-integrated engineering system?
- Is the CAX framework approach as effective as expected with adequate demonstrations on a physically developed prototype?

The significance of studying these problems is reflected in several aspects, which will be elaborated in the following sub-sections.

1.2.3. Development of a Prototype with a Long-term Objective for Industry Application

The most apparent value of this research is that the result of the prototype may be used by the industry with some further developments according to the methodologies presented in this thesis and some other widely-known technologies. The scenarios supported by the developed system are not purely imaginary like those proposed by many other researchers, *e. g.* Urban *et al.* (1996), Qiang *et al.* (2001), Gerhard *et al.* (2001), Wang & Zhang (2002), Li *et al.* (2004), *etc.* They are abstracted from a real-

world complex product development process of a type of sheet metal products using progressive dies. The implementation decisions are made by using the latest information technologies which are both challenging and easily available. Compared with existing systems in the prototype-focused area, like the NUS IPD system (Cheok & Nee 1998a, b; Jiang *et al.*, 2000; Zhang *et al.*, 2002), it has many advantages. Firstly, the system is more flexible with more function modules (engineering tools) easily integrated into the system. The scope is not limited to all aspects of die design, *i.e.*, product feature modeling, unfolding, nesting, die operation planning and die configuration. Die manufacturing, *i.e.*, die parts process planning and NC codes generation, can also be easily integrated into the system. Secondly, the data management and process management functions based on the CAX framework methodology are newly created and embedded into the system. Product data integrity has been improved with easy access and without data redundancy based on a shared product database, which also serves as a communication medium for the engineers involved. Engineering activities to drive product realization from upstream stage towards downstream stages are easier to master for the end-users and less error prone with maximum cross-process automation. Thirdly, the single-user operation mode has been extended to a multi-user one, which allows data/knowledge exchange and sharing among the engineering team-members and supports cooperation among participating engineering tools working in different computers that are geographically dispersed across the enterprise. Fourthly, the CAX framework provides intelligent facilities to upgrade the product database to a knowledge base, so that the design knowledge embedded in the past product models created by any team-members using the relevant engineering tools is naturally captured and easily retrieved when needed to help the users to interrogate solutions for the current case. This also lays a foundation to use

advanced Case-Based Reasoning (CBR) technology to further enhance system intelligence.

1.2.4. Theoretical Value of the Present Research

When developing an integrated engineering environment, the challenges are numerous and the solutions are diverse. The current study is theoretically important in this broad area in that it is not just another novel example system with some new technologies (such as the Internet technology) adopted and with plain implementation decisions described at a detailed level. Priority was firstly given to capture the underlying common principles to meet the challenges in a large range of engineering environments similar to what the case study has indicated. The proposed notion of feature-driven engineering processes abstracted from the case study may improve the understanding on how a category of complex engineering processes are decomposed into sub-processes and in what way these sub-processes interrelate with each other. The formulation of the captured principles and the success of using them in developing a concrete system may imply that a new system integration pattern, the CAX framework, has been discovered to enrich the current system design theories. Like other integration patterns, such as the Multi-Agent System (MAS), the CAX framework pattern provides reusable architecture templates to address recurring problems and implementation hints to ensure a strong likelihood of achieving a successful solution when it is tailored to any other applicable context. Emulated from its parents, the CAD framework, the CAX framework technology itself has a number of advantages as a system integration methodology. Benefits include cutting down product development time, increasing performance and quality of products under engineering, and making the development process less error-prone.

1.2.5. Other Potential Application Areas of the Research

As conceived and tested in this research, the concept of feature-driven engineering process and its integration approach in a CAX framework have been intentionally biased to the development of an integrated engineering environment for sheet metal products using progressive dies. However, they may be also valuable outside this important area. A variety of product development cycles can be characterized as a feature-driven process and thus the current approach is applicable to them. For example, the development cycle of injection-molded products is very similar to that of the sheet metal stampings and also needs a set of feature modeling and mapping tools. For another example, most of Integrated Product and Process Design (IPPD) systems have a structured process pattern resembling the feature-driven engineering process model and thus might be promising application areas of the CAX framework approach. Here is a demonstrative IPPD scenario in a modern manufacturing environment: feature-based modeling of a car being integrated with another design automation tool to design car robotic arms, which are controlled to assemble the car. Another IPPD scenario described in the aforementioned literature review section, the integration of three tools (components): CAD, a process planner and an inspection planner (Marefat *et al.*, 1993), may also benefit from the CAX framework approach.

If removing the limitation to use feature models as the underlying operation models of the participating tools in the integrated engineering environment, the concept of the CAX framework would have a wider application scope. It would be open to any types of engineering tools, including CSCW and many other new generation ones focusing on distributed concurrent engineering. It is noted that these new generation tools are

currently conceived and tested as a subsidiary functional module outside the mainstream product design and development environments and need to work out a way to coherently integrate them with others sharing a common engineering task (Mervyn *et al.*, 2003). The CAX framework can play the role of a software infrastructure that provides a common operating environment for all distributed concurrent engineering applications involved. Therefore, it may be able to fulfill the above need given that the underlying facilities are adjusted accordingly.

1.2.6. Research Scope and Overall Approach

As stated, this research concerns itself with the development of a network-integrated and distributed engineering environment using the CAX framework technology, a new concept derived from the CAD framework that is originally found in the area of EDA. The application context is the full product design and development cycle of mechanical products which have a feature-driven process model. To demonstrate the conceptual feasibility of this approach, the characteristics of the intended application context were investigated to make a comparison with that of a typical application context of a CAD framework. Instead of identifying all aspects of the analogy between them, the focus was placed on characterizing the relationship among a group of CAX (CAD) tools. It was revealed that the most important impetus underlying the research and application of the CAD or CAX framework is its ability to integrate a range of engineering tools which have a logically centralized coordinator. Similar to the CAD framework, the CAX framework is scalable and can be configured to encompass a range of functional components and thus can be allotted various roles. However, this research was mainly limited to its three basic roles: engineering data repository, engineering data manager and engineering process manager. Advanced functions, such

as knowledge repository support, cooperative engineering transactions, reusable CSCW-like services, *etc.*, were mentioned wherever appropriate but not thoroughly studied.

According to CAD framework principles, three well-formulated steps are recommended to be taken to develop a CAX framework-enabled engineering environment (Wolf 1994). The start point is to derive a model of the targeted engineering environment. This model provides a vocabulary of well-defined terms, and thereby a context for functional specifications. The second step is to identify the logical structure of the framework which indicates the details of the framework functions including the unspecified ones of the framework services. The final step is to complete the definition of the integrated engineering environment at the physical level. Many decisions are made at this step, which has no special principles to follow. This three-step pattern which allows iterations has been sequentially followed to initialize a practice to develop a prototype system at the beginning. However, the sequence was not eventually used to formulate the current development efforts and neither recommended to other interested researchers because of its absence of incorporation of the OO principles. A two-step strategy is used in this thesis. Firstly, a “skeletal” CAX framework up to the physical level is developed. Afterwards, the development effort is biased to concentrate on the most creative and challenging aspects: modeling and analyzing the desired engineering environment to generate an adequate schema for the management database and devise the required operations on the data. It is found that this two-step strategy is more natural for system developers and probably helpful in reducing unpleasant iterations before a satisfactory system specification is achieved given that the CAD framework principles have been acknowledged in advance.

To demonstrate the approach to develop an integrated engineering environment using the CAX framework technology, a full case study was conducted in the area of sheet metal products using progressive dies. A set of selective demonstrations was designed to assess the effectiveness of the approach. In summary, while there are many perspectives to view the CAX framework-enabled engineering environment with each one emphasizing particular aspects of the architecture, this research explored the system modeling perspective on the abstract level and the implementation perspective on the physical level using a case study to exemplify all the details involved.

1.3. Terminology Statement

Beginning from a broad scope in the development of an integrated and distributed engineering environment, the focus of this research was fine-tuned to a fully new topic, integrating distributed feature-driven engineering processes in a CAX framework. Viewing some complex engineering processes as “feature-driven” ones is an elegant way for processes integration. The underlying idea stems from “data-driven, information-driven or model-driven” where “model” now specifically refers to feature-based model. The CAX framework is a key concept for this research topic, and probably requires a precise definition before presenting the details of this approach.

In a broad sense, according to The Merriam-Webster Dictionary,

“A framework is a skeletal, openwork, or structural frame.”

It is noted that almost all the integrated and distributed engineering environments have their own framework as an architectural skeleton, on which the full system is based. Some of them are obviously denoted while others just obscurely contained in the system. In both cases, the framework for a specific integrated engineering environment usually plays a constricted role to act as an internal expedience personally-owned by the system developer for partitioning the domain, layering the architecture and fully specifying the system. Since no generic framework design principles and patterns are investigated and made available before the system design, all works are done from scratch and thus the development practice is often slow and unpredictable.

The framework in the context of the current topic of “CAX framework” has a small difference with the above concept. It has semantics of an OO application framework in software engineering, where a precise definition is given as following:

“A framework is a reusable, ‘semi-complete’ application that can be specialized to produce custom applications (Johnson & Foote 1988).”

The primary benefits of OO application frameworks stem from the modularity, reusability, extensibility, and inversion of control they provide to developers (Fayad & Schmidt 1997). While the framework in this sense can be classified by their scope into three categories, system infrastructure frameworks, middleware integration frameworks and enterprise application frameworks (Fayad & Schmidt 1997), the framework in the current study falls into the level between the middleware integration framework and the enterprise application framework. It defines a semi-complete application that embodies engineering domain-specific object structures and

functionality. Components within it work together to provide a generic architectural skeleton for a family of related applications and the complete applications can be composed by inheriting from and/or instantiating these components.

Therefore, the current CAX framework is not a spontaneous “by-product” throughout the course to develop an integrated engineering environment. It is a conscious effort to capture the common framework knowledge which may be recurrently applied in different context and encapsulate volatile implementation details behind stable interfaces. As has been mentioned above, this idea is inspired by the CAD framework in the area of EDA. The authoritative definition of CAD framework is given by the CAD Framework Initiative (CFI), the international consortium developing framework standards (CFI 1990b):

“A CAD framework is a software infrastructure that provides a common operating environment for CAD tools.”

Similarly, a CAX framework is a software infrastructure that provides a common operating environment for CAX tools. Various roles can be allotted to the CAX framework depending on the way in which it is specified. It can be basically exploited to integrate dispersed CAX tools for the tool users. It can also be exploited to achieve more effective collaborations among these users. In this sense, from the methodological perspective, the CAX framework is an enabling technology, which functions as the centric concept of the proposed integration approach for development of a network-integrated engineering environment. On the other hand, from the structural perspective, the CAX framework in a physical CAX framework-based

engineering environment is an integration tool or collaboration tool, which co-works with the surrounding CAX tools.

14. Thesis Organization

The remainder of this thesis is organized as following. Chapter 2 provides a comprehensive literature review on the existing efforts to develop an integration infrastructure for complex engineering systems so as to reinforce the above statements on the open issues for integrating feature-driven engineering processes. Chapter 3 characterizes an engineering process from the perspective of feature-driven engineering. Process decomposition, dependency relationship identification and adequate design transaction models are comprehensively addressed. Chapter 4 presents an overview of the CAX framework-based integration approach. A “skeletal” CAX framework is incrementally derived from a small set of high level primitives. Chapter 5 depicts how the “skeletal” CAX framework is enriched with the product data integration functions. A versioning control and configuration management model is presented. The corresponding operational issues are also addressed. Chapter 6 depicts how the “skeletal” CAX framework is enriched with another important function, the process management function. The finally-obtained system is a network-integrated engineering environment using an integration approach which is both data and process-centric. Process management mechanism design and process modeling are emphasized and the overall information model including a UML sequence diagram is described. Chapter 7 presents the results of a demonstration session working on the prototype system. Chapter 8 summarizes the contributions made by this study and outlines areas of future work.

CHAPTER 2

LITERATURE REVIEW

This chapter presents a general review on past and current researches into system integration from design to manufacturing. Although a huge body of literatures can be found having relevance to this topic, it is far from being able to be treated as a formal discipline which has a consensus amongst its community on its research directions, scope, issues involved and reference paradigms. The objective of this survey is to gain insights into what kind of new research efforts may truly contribute to this area with both theoretical and practical values. Therefore, conclusions drawn from this survey may be repetitively used somewhere in Chapter 1 or the chapters following this one. The survey itself includes a historical perspective, some aspects significantly affecting integration and a suite of sample integration architectures.

2.1. A Historical Perspective on System Integration from Design to Manufacturing

Maybe to some researchers' surprise, all activities from design to manufacturing were seamlessly integrated by nature in the beginning according to Cross (1989). Both design and manufacturing, if these terms were used by the people in that era, actually referred to the same activity to physically fabricate an artifact. Craftsman would design as they manufactured and manufacture as they designed (Jeremiah 18:4) (Mowchenko

1996). Separation of design and manufacturing into two islands of activities occurred when sophistication both on the design side and the manufacturing side progressed to such an extent that documenting and detailing the specifics of a design should be completed before manufacturing. Further, with the introduction of computer-based systems, more task-specific tools would be adopted to perform partial design or manufacturing activities, and more activity islands may be found in an entire development cycle from design to manufacturing.

Wherever there were separate activity islands, integration efforts would be devoted to filling the gap sooner or later. Before computer-based tools were introduced for these activities, it is their cooperation and collaboration efforts between design engineers and manufacturing engineers that fully took this responsibility by translating and re-implanting information encapsulated in designs into corresponding manufacturing specifications or the manufactured artifacts. Soon after the first computer-aided design tool “sketchpad” was produced by Ivan Sutherland at MIT in 1962, a team at General Motors Research Laboratories developed a system which not only displayed shapes on a screen, but also linked this information to NC controlled machines. This led in 1964 to the construction of the first CAD/CAM system called DAC1 (Design Augmented by Computer) (Black 1996). Therefore, even at the early stage of the evolution of computer-aided systems, it was recognized that a unique system-level capability, integration from design to manufacturing, may be leveraged to alleviate engineers’ workload if adequately addressed. One of another earliest integrated systems reported in literature is ROMAPT (Chen 1982), which integrated the CAD system ROMULUS and the NC system APT. Throughout the 1980’s and 1990’s, an explosion of research interests into system integration from design to manufacturing can be observed, which leads to a special term, CIM (Computer-Integrated Manufacturing) or CAD/CAM

being used to collectively document all efforts along this line. According to its formal definition, CIM also concerns integration of production functions (Singh 1996), but the main issues addressed are almost identical.

Although system integration from design to manufacturing has advanced for over 30 years and researches in this area are still active, it is difficult to clearly stage how it has been evolved. This is probably because there are no breakthrough methods out there although some methodological questions may be very tough. Most of the researches are devoted to broadening its application scope with introduction of a new (type of) system fitting to a particular new context with few radical methodological improvements. Few researches are contributed to making obvious progress in its theoretical and technique depth. While it is difficult to gain a clear picture of its evolution by defining a set of representative indicators as can be found in describing the evolution of CAD, one can notice that some strategies or core technologies with much formulation have strongly affected its evolution. Taking a look into some details in these aspects not only helps to trace the technical evolution but also helps to capture appropriate metrics to measure the value of a particular system integration effort. The strategies should not be viewed as evolution indicators because overlooking any one of them in a specific application does not necessarily bring about an immediate inferior solution.

2.2 Some Aspects Driving System Integration from Design to Manufacturing

Six significant aspects that drive system integration from design to manufacturing are presented in this section. They are information modeling, concurrent engineering, intelligent integration, data integration, process integration and object-orientation.

- Product and process modeling

Information modeling, including product and process modeling, is probably one of the most useful analytical techniques employed to overcome integration barriers in data exchange and sharing amongst design and manufacturing software systems. The application of modeling helps in better understanding and easier handling of the modeled system by dealing with the purposely selected features. Respectively, product modeling develops data models and process modeling develops activity models. A data model defines the data elements and their relationships. An activity model describes a process activity and its sub-activities, as well as the data associated with the activity (Algeo *et al.*, 1994).

Two groups of researches may be viewed as the origins for all others in this aspect. One is known as the IDEF (Integration DEFINition) technique series including IDEF0, IDEF1 and IDEF2 modeling methods developed by the U.S. Air Force ICAM Program during the 1976 to 1982 timeframe (U.S. Air Force 1981). The other is known as the STEP (STandard for the Exchange of Product model data) standard suite (ISO 1994). The IDEF series includes an activity modeling language IDEF0 and an ER (Entity Relationship) modeling language IDEF1X which is an extension of IDEF1 and many others without receiving much concern. The core of the STEP standard suite is not restricted to information modeling methods themselves but have an ambitious aim at creating normalized object models to allow any manufacturing applications to share product data if semantically possible and desirable. The activity modeling method it used is IDEF0 and the data modeling methods it used is EXPRESS, a modeling

language of its own. EXPRESS and IDEF1X can be viewed as functionally identical but the former seems more popular probably because its modeling power is stronger.

Two types of data sharing and thus application interpretability for integration are supported by STEP. One is within the same application areas but operating on heterogeneous platforms and the other is across many application areas in the entire product life cycle from design to manufacturing (Zhang *et al.*, 2000). While STEP is quite successful in the first type of sharing through neutral representations of the product models across heterogeneous computation platforms, for example sharing CAD models between CATIA, Pro-E or any other CAD platforms, few successful real-world stories can be found for STEP to be implemented with the second type of sharing across application areas even on the homogeneous platforms. This is probably because STEP proves to be too unwieldy (Hillebrand *et al.*, 1998) in this aspect due to the expensive normalization which makes the developed models harder to process (Hardwick *et al.*, 1996).

However, the integration philosophy underlying the STEP standards has been widely used. In general, from the perspective of system analysis based on information modeling, implementing integration from design to manufacturing begins from its opposite side: decomposing the process into manageable sub-processes in terms of activity modeling. Examining the interoperability between the corresponding intermediate product data models for each sub-process is the next task and a global integration model capturing the common semantics shared by all participating intermediate models is probably required (Dhamija *et al.*, 1997). IDEF0 has become the *de facto* activity modeling language and EXPRESS is a popular data modeling but

far from the only and best option. Most researchers select UML (Unified Modeling Language) as the data modeling language instead of EXPRESS probably because of UML's broader acceptance in the software engineering world. It is yet noticed that the modeling practice is often more important than the language chosen (Lee 1999), because any one of them can be mapped to the other (Arnold & Podehl 1998). This thesis will use IDEF0 to perform process decomposition and activity modeling and UML to perform data modeling.

- Concurrent engineering

Technical advancement of system integration from design to manufacturing has been an evolutionary process attributed to an increase of awareness rather than a revolution driven by certain technology leaps. For example, integration was mainly perceived in its early days as a means to smooth the information transition from one lifecycle function module to the next for the purpose of reducing user effort and increasing consistency (Singh 1996; Black 1996). Increasingly with more experimental solutions to the problems in this aspect introduced to and acknowledged by industries, the limitations of the solutions were also recognized and more profound understanding to the tenet of integration came out: integration is much more than mere coupling of processes, or information flows between them, what one can call module-module interaction (Singh 1996). On the other hand, integration facilities should also allow effective user-user cooperation and user-module interaction. One of the most important management and engineering philosophies which have fostered such an increase of awareness of integration tenet is Concurrent Engineering (CE).

As “the art of decomposing a complex serial task into smaller, relatively independent tasks that can be executed in parallel” (CFI 1990a), CE advocates maximum concurrency of engineering activities involved in an engineering process for which an integrated system is about to be constructed. Ensuring that an integrated engineering system be able to fully support the CE strategy has been widely accepted as one of essential procedural to specify an integrated system. Some integrated systems, such as the CONCERT (CONCurrent Engineering support) environment (Hanneghan *et al.*, 1995, 1998), may even be called a CE system firstly and any others next. The integration approach proposed in this thesis will also be linked to an examination on how CE is supported.

- Knowledge-based system / intelligent integration

Historically, knowledge-based systems contributed a lot to intelligent design and manufacturing mainly by providing new mechanisms to develop individual task-specific CAX tools to automate a large proportion of routine design and manufacturing tasks. It also has significant relevance to system integration from design to manufacturing in that there had ever been a great shortage of implementation means for system integration and knowledge-based system framework. The Blackboard Architecture (Hayes-Roth 1985; Nii 1996), offered a type of technical possibility. Although there is no agreement on what should be included in the integration infrastructure for a range of design and manufacturing applications to realize a truly integrated engineering environment, some basic functions would be inevitably involved. Examples of such functions include a unified and coherent user interface able to instantly navigate around the real-time participating applications, common product database shared by the participating applications, some context-sensitive

control logic to smooth the cooperation between corresponding applications, *etc.* Before the prevalence of the multi-process operating systems like the Microsoft Windows[®] and the OO programming tools like C++ and Java language, even realizing the coexistence of two applications in a single session is a big problem. Programming the control logic with some basic heuristic reasoning capabilities is also difficult for software developers using traditional procedural programming languages such as FORTRAN and C. It is natural that the knowledge-based system framework, the expert system shell, was selected by many researchers to function as an intermediary that operates in a loose integration fashion with the surrounding applications. One of the earliest works justifying the expert systems' capability for system integration was contributed by Madison *et al.* (1988), who proposed an expert system translator to link the CAD and CAM and help them speak the same language. Another example system used an old-fashioned OO programming environment (Smalltalk-80, Version 4.0) to realize the integration of three tools (components): CAD, a process planner, and an inspection planner (Marefat *et al.*, 1993). These components all share a common database that acts as an intelligent integrating agent. Note that the underlying integration mechanism through the expert system framework or the shell is actually attributed to a shared "blackboard" although this term may not be explicitly mentioned.

The terminology of blackboard architecture based on the knowledge-based shell makes its integration capability more formally documented, easily understood and broadly recognized. Participating applications to be integrated are nothing more than special knowledge sources in the blackboard-based integrated engineering environment, which consists of a blackboard, an inference engine-enabled agenda controller and any number of knowledge sources. A design database used to store all the dynamic design

data produced by the surrounding applications is a *de facto* knowledge resource and thus the product data management function is always provided by such environments. Numerous integrated engineering environments based on the blackboard architecture have ever been reported. Some examples are listed in the following. Megale *et al.* (1991) introduced a CAD-CAM integration environment based on the blackboard architecture. Palani *et al.* (1994) developed an intelligent design environment which integrates a sheet metal part CAD module, an FEA module and a design evaluation module using the blackboard architecture. Fagan (1994) presented an integrated environment consisting of a myriad of computer-aided engineering design and analysis applications for engine crankshafts utilizing the blackboard approach as an implementation tool. Srihari *et al.* (1994) described a blackboard-based process planning system, which integrates a planning sub-system performing static process planning tasks and a dynamic information processing sub-system, for the surface mount manufacture of PCBs (Printed Circuit Boards). Roy *et al.* (1995) developed a knowledge-based process planning system using the CLIPS V6.0 expert system shell to integrate with a feature-based design system working on the CAEDS solid modeler through the blackboard architecture. Hayes (1995) described CHAMP, a conceptual architecture designed to support the task of passing information from CAD systems to CAPP systems. The proposed architecture facilitates CAD/CAPP integration through shared blackboards.

One of the common features that can be found in all the above blackboard-based integrated environments is that they consist of at least one knowledge-based application which relies on the common expert system shell which functions as the basic platform for the blackboard. Today's knowledge-based design and

manufacturing applications prefer to be compact, with the knowledge imbedded interiorly owing to the strong expressing and reasoning ability of the OO programming languages such as C++ and Java. No exterior expert shell is strongly needed. Further, the multi-process operating system such as Microsoft Windows[®] popularly available on PCs (Personal Computers) makes easy coexistence of multiple applications and resources. Therefore, the significance of the contribution of the knowledge-based systems towards system integration is reduced. This observation has been reinforced by the author's experience gained when participating in the IPD (Intelligent Progressive Dies) project (Zhang *et al.*, 2002) initiated by National University of Singapore and the Institute of High Performance Computing. As an integrated and intelligent toolkit for the design and manufacturing of progressive dies, IPD in its original version realizes the integration of a range of modules (applications) including a product feature modeler, an unfold, a layout planner and a die configurator through an in-house knowledge base shell. The underlying architecture is compatible with the blackboard concept. However, the new version of IPD has removed the shell because the limited integration capability provided by the shell can almost come free in the Microsoft Windows[®] platform, and the knowledge bases as well as the inference engine provided by the shell to any individual module can be directly coded into that module.

Nevertheless, the contribution made by the knowledge-based systems in the evolvement of system integration from design to manufacturing cannot be overlooked. For example, it has helped in improving the understanding on the integration problem, specifically what may be required to be included in a concrete integration implementation. It also helped to foster a new integration paradigm, the client-

knowledge server architecture (Eriksson 1996) corresponding to the recent hot research atmosphere for distributed design and manufacturing owing to the emergence of the Internet-centric technologies. In this client-knowledge server architecture, the inference engine and knowledge bases (shared blackboard and basic knowledge sources) are located at a server computer, and interfaces are exported on demand via network connections to client computers where a common GUI (Graphic User Interface) application and purposely selected functional design/manufacturing applications are running. Integration in this architecture may be between two functional applications of different types or even the same type but running on different sites. The development efforts using this integration paradigm includes a computer-based design system proposed by Sriram & Logcher (1993). It provides a shared workspace, *i.e.*, a blackboard, where multiple designers work in separate engineering disciplines. In this distributed and integrated environment for computer-aided engineering (DICE) program, an OO database management system with a global control mechanism is utilized for coordination between distributed users and applications. Another such kind of effort is made by Zha & Du (2002) who developed a design platform with client-knowledge server architecture for collaborative design of Microelectromechanical systems (MEMS) through concurrent integration of multiple distributed knowledge sources and software. Although the client-knowledge server architecture realized some crucial integration functions in the distributed environment, these integration functions are far from sufficient for realizing a comprehensive integrated engineering environment consisting of a set of distributed CAD/CAM applications. This is probably one of the reasons why very few researchers selected the client-knowledge server architecture as a means to achieve more sophisticated integration architecture. They would rather develop distributed and integrated

engineering environments of their own on a project-by-project base from scratch using basic distribution and OO technologies (see the next section).

- Product Data Management (PDM) / data integration

PDM, a technology developed for the integration of CAX systems to manage product data centrally (Conaway 1995; Norrie 1995; Anonymous 1998; Fan 2000), is probably the only formulated and industrialized technology that has been being widely employed as an effective means to solve some integration problems involved in product development process from design to manufacturing. PDM can be exploited in the narrow sense or in the broad sense by the integrated engineering environments. However, it is unfortunate that in most cases, PDM is only stressed in the narrow sense and its influence on system integration in the broad sense has been inadequately addressed in the community of PDM.

PDM in the narrow sense refers to a PDM application or PDM system, an off-the-shelf software tool. PDM in this sense can be traced back to the early 1980's when many large corporations, often the leaders in the engineering-manufacturing industry, found their efficiency severely downgraded by the poor management of the huge bulk of electronic product lifecycle-related information using the traditional paper-based means (Liu & Xu 2001; Xu & Liu 2003). Driven by the ever-growing potential market of efficient product data management methodologies, several generations of commercial PDM systems have been introduced to the manufacturing industry and a multitude of PDM products are available on the market. While a PDM system is a crucial tool for the management of the large amount of data generated by computer applications to ensure that the right information is available to the right person at the

right time and in the right form throughout the enterprise, it can also function as an intermediary to integrate a set of interrelated applications like the above-mentioned intelligent blackboard. Iuliano (1995, 1997) described in detail how a PDM system (Adra Systems' Matrix™ V3.0) is used to implement an integrated plug-compatible environment consisting of a CAD application (Parametric Technology Corporation's Pro-Engineering™), a generative process planning application (Technomatrix's ICEM™ Part) and a suit of manufacturing simulation applications (Deneb Robotics' Igrip™, Quest™ and Virtual NC™). However, the integration capability provided by PDM applications in this way is very restrictive. This is because the PDM application is primarily targeted to interface with end-users rather than the participating CAX applications involved in a product development process. Being utilized as an intermediary to integrate CAX applications is only its secondary function. It has no knowledge of the existence of the surrounding CAX applications to be integrated and the environment constructed in this way is a more loosely integrated one than that constructed using the blackboard architecture. Much manual effort from the end users is still required to make the full system work coherently and effectively.

PDM in the broad sense refers to the PDM function which is a set of data integration decisions as a part of the development practice to realize a large integrated toolkit consisting of a myriad of CAX applications. PDM in this sense is far more important for system integration from design to manufacturing than PDM in the narrow sense because the PDM function is almost mandatory for any integrated engineering environment. Incorporation of an optimal PDM function within an integrated framework makes it possible to realize the maximum degree of integration for a given non-integrated process with the minimal artificial efforts required, especially in the

aspect of data integration. Fortunately, some researchers outside the conventional PDM community made considerable contributions in this aspect. For example, researchers from Arizona State University studied the mechanism to use an OO database system as a Shared Design Manager (SDM) to provide a blackboard for communication among CAD tools (Urban *et al.*, 1996). SDM uses a STEP product model as a global conceptual view of data and is very flexible in the configuration of the design environment and in establishing communication. Based on the success of SDM, two other components, Integrated Product Database (IPDB) and a set of Data Access Interfaces (DAI) for each application type, were added to the environment. An Integrated Product Design Environment (IPDE) is then formed which allows CAD/CAM/FEA programs to share data dynamically and operate coordinately (Shah & Urban 1998). The data modeling aspect of IPDE was discussed by Liang *et al.* (1999). The development result and demonstration of the IPDE was also elaborated especially in the aspect of database framework (Urban *et al.*, 1999a). The feasibility of the use of Common Object Request Broker (CORBA) tool in the existing IPDE was further studied (Urban *et al.*, 1999b). The feasibility study on employing the Oracle[®] 8 object-relational data model to re-implement the database kernel in IPDE was also conducted by using the STEP EXPRESS conceptual modeling language (Urban *et al.*, 2000).

Another representative effort dealing with data integration was made in the context of the SUKITS project (Schwartz & Westfechtel 1993). The SUKITS project is targeted at a *posteriori* integration of existing CAX applications into an integrated CIM system – the CIM Manager. The CIM Manager manages versioned, interdependent documents (including CAD designs for different purposes, CAPP plans, NC programs, and FEA

simulation results, *etc.*) which are combined into configurations. PDM in the SUKITS project is referred as product management, based on which process management and resource management were also enabled by the CIM Manager (Westfechtel 1996; Westfechtel 2000). The SUKITS architecture follows the client-server paradigm with CAX applications residing on the client computers and management tools provided by the server machines. Since the SUKITS prototype was heavily based on the software and services originally devoted to a different application domain, software engineering, its conceptual framework did not neatly address system integration issues for engineering processes from design to manufacturing. However, this research, as well as those presented in previous paragraph and many other similar ones (Rundensteiner 1993; Wang *et al.*, 1993; Bounab & Godart 1998; Karsai & Gray 2000; Roller *et al.*, 2002a, b), laid a moderate foundation for understanding the PDM function-enabled system integration mechanism and making basic implementation decisions for designing an adequate integration architecture and addressing relevant issues.

To summarize, the PDM functional module in an integrated engineering environment possesses a shared, persistent data vault or database engine for multiple CAX applications and manages “data about data” or meta data of the physical product data in an official and semantically unambiguous form. It contains the data object identity, pointers to product data, the relationships between product data, product structure relationships and administrative data. The meta data can be organized in multiple abstract levels and has a complicated information structure, hence data modeling should always be performed to precisely define its schema. One of the most important issues for data integration is probably versioning control and configuration

management. The state-of-art in this aspect will be presented in Chapter 5 when the author's own data integration solution is proposed.

The PDM in the broad sense plays a more significant system integration role than that in the narrow sense in that the PDM functional module in an integrated environment is at a higher level and monitors the participating CAX applications but the PDM system when being coupled for integration is parallel with others.

One of the prominent features of PDM-based integration architecture that is different from the intelligent blackboard architecture is that the former is targeted at distributed and multi-user environments from the beginning of its emergence although the technologies it used before were quite distinct from that at present. Therefore, contrary to many researchers' assertions, data integration in a distributed environment via network-enabled communication is far from a new topic. The effectiveness of some historically-proposed integration approach may still hold. New valuable research efforts may be required to carefully identify and better characterize specific application contexts in which the integration philosophy is to be enforced, derive a decent integration approach based on formulating and unifying historical achievements and experiment with more effective distribution technologies.

- Workflow Management (WM) / process management / process integration

The best system integration solution for a specific engineering process, if exists, is relative because the technologies in use would be continuously evolving. However, the primal thrust to and the ultimate goal for integration seems to be permanently located at the improvement of process automation, minimization of interruption and reduction

of the participants' error-proneness. In the domain of EDA, one can observe two process control approaches indicating two levels of design automation (Schrmann & Altmeyer 1997). For the tool-based approach that is generally implemented with the early design systems, the designer is completely responsible for his design without having computer-based support for supervising the design process. For the task-based approach which is able to handle the increasing design complexity beyond the suitability of the tool-based approach, the shared data management function in the integrated framework is complemented with a computer-aided process management service, which can off-load the designer's highly-demanding process management efforts. This observation implies that a similar process management service may be desirable to be devoted to design automation and process integration for the mechanical product development process from design to manufacturing.

As Hillebrand *et al.* (1998) pointed out, a CE-compatible old-fashioned data integration mechanism based on a logically centralized information base is a necessary, but by no means sufficient precondition for the successful integration of a collaborative engineering process. Most engineering processes should be able to be characterized by following certain established patterns if the process knowledge is adequately captured; hence one should be able to superimpose suitable process models on the shared database system to supplement the traditional PDM mechanism. Unfortunately, very few researches have been devoted to clearly illustrating how the process models are specified and how such models are superimposed, or to simply put, the conventional data integration function is augmented with process management assistance.

It can be noticed that some PDM systems began in the 1990's to make use of workflow, a concept primarily connoting a highly structured, repetitive process in a business management application (Georgakopoulos *et al.*, 1995), to automate project management processes (Ramanathan 1996; Fan 2000). Likewise, researchers like Heimann & Westfechtel (1997) suggested the incorporation of the WM mechanism to integrate the activities performed by a set of CAX applications in a sequence following some rules. A WM system views a business process as a workflow consisting of a range of tasks (activities) with a predefined execution sequence and a group of predefined execution constraints. It allows the end-users not only to execute tasks defined in the workflow, but also define and modify the workflows themselves: the workflow is also a product on which participants are working (Heimann & Westfechtel 1997). Process management support for some complex product development processes are both desirable and feasible because they are dramatically repeatable at the activity level like business processes and potential to be modeled as workflows to incorporate the WM technologies and facilities into the basic data integration infrastructure (Ramanathan 1996; McClatchey *et al.*, 1998). This suggestion was, however, challenged by Hillerbrand *et al.* (1998) who argued that WM imposed too tight a discipline on the sequence of process steps (activities). The tight discipline can, however, be loosened in many ways such as using the intelligent agent-based approach to achieve dynamic process adaptability, which allows changes to the workflow during execution (Kim *et al.*, 2001). The genuine obstacle to obstruct the use of the mature WM technologies and facilities is probably attributed to the fact that the engineering process is usually semi-structured and hence hard to handle because it is not clear how to balance flexibility and control (Westfechtel 2000). To tailor even the most amiable WM system to manage a specific engineering process is consequently almost of the

same difficulty as to develop a completely new process management system with a unique process definition model and process execution engine of its own. Therefore, some researches would rather develop process management methodologies, *e.g.*, those proposed by Hillerbrand *et al.* (1998) and Zhang & Luttermelt (1995), outside the conventional WM conceptual framework. Even the workflow system within the integrated product development environment in the context of projects like the SUKITS project (Heimann & Westfechtel 1997) was quite different from the conventional WM systems.

In summary, the presence of process management services is an important factor in the consideration for the development of system integration infrastructures for a complex engineering process. On the other hand, the design of a conceptual framework for process management and process integration is still an open research issue. This thesis will identify process management functions based on the space-state model, a generic mathematical model underlying all physical processes, and develop process management mechanisms based on the design flow concept, which was widely used in CAD framework in the domain of EDA.

- Object-Oriented methods and distributed object technology

All proposals to integrate a set of separate but logically related CAX applications into a unified and coherent engineering environment have been and will continuously be heavily dependent on the information technologies (IT) currently available. Many system integration initiatives did not finally find industrial applications because the advances in computing abilities have been so great that expectation has, in some case, run a little in front of reality (Black 1996). Of all IT advances that occurred during the

last two to three decades, OO methods as well as their successive distributed object technology are probably one of the most significant to affect the integrated system developers' philosophy and practice. With the OO methods, it is outdated to utilize some specific intelligent technologies, such as the blackboard architecture to implement system integration. The following paragraphs examine in a wider scope the impact of OO methods and distributed object technology on the development of network-integrated engineering environment.

OO is a software system developing method that uses abstraction with objects, encapsulated classes, communication via messages, object lifetime, class hierarchies, and polymorphism. It is a well-established and effective way to develop software, and is certainly the dominant method used to develop major software systems today. Note that application of the OO method is not just a practice of writing software programs using OO programming languages such as Java and C++. It actually covers the complete software development process—analysis of the problem, design of a solution, coding, and long-term maintenance. It is even said that any programming language can be used to write OO programs (and it has been done with C), and of course, a true OO programming language makes it radically easier (Wampler 2001). Advantages of using the OO approach to analyze and design CIM system architectures was recognized even in the late 1980's as reported by McFadden (1989). However, the comprehensive use of the OO methods to solve system integration problems in comparison with the natural language- and process-oriented methods came at the beginning of this century (Law & Tam 2000). The next paragraph presents a full picture on how an elegant, easy-to-understand integration infrastructure design can be produced using OO methods given that a corresponding integration mechanism exists pre-conditionally.

One of big payoffs that OO can lead to is that the individual objects within a system can be implemented and tested separately. On the other side, an integrated engineering environment including the integration infrastructure and the participating CAX applications is viewed as an OO system for the purpose of embracing OO methods in this case. If the interfaces between the CAX applications with the integration infrastructure are defined, the system-deductive principle makes the integration infrastructures to become another OO system independent of all the component CAX applications. In the simplest terms, designing an OO system consists of identifying which objects the system contains, the behaviors and responsibilities of those objects, and how the objects interact with each other (Wampler 2001). More specifically and according to Singh (1996), in designing the integration infrastructure system using the OO paradigm, a systematic procedure can be used as shown in Fig. 2.1. The first two steps in this figure are probably most challenging and a lot of analyzing efforts, like those described in the following chapters of this thesis may be required. The traditional functional and structural analysis concepts based on the top-down and/or bottom-up methods may be still obligatory to be assimilated. Incorporation of some intermediate functional abstractions, such as software components located between the global system level and the primitive object level may be especially supportive. In short, OO methods always mean good traceability from the system requirements to the final physical implementation.

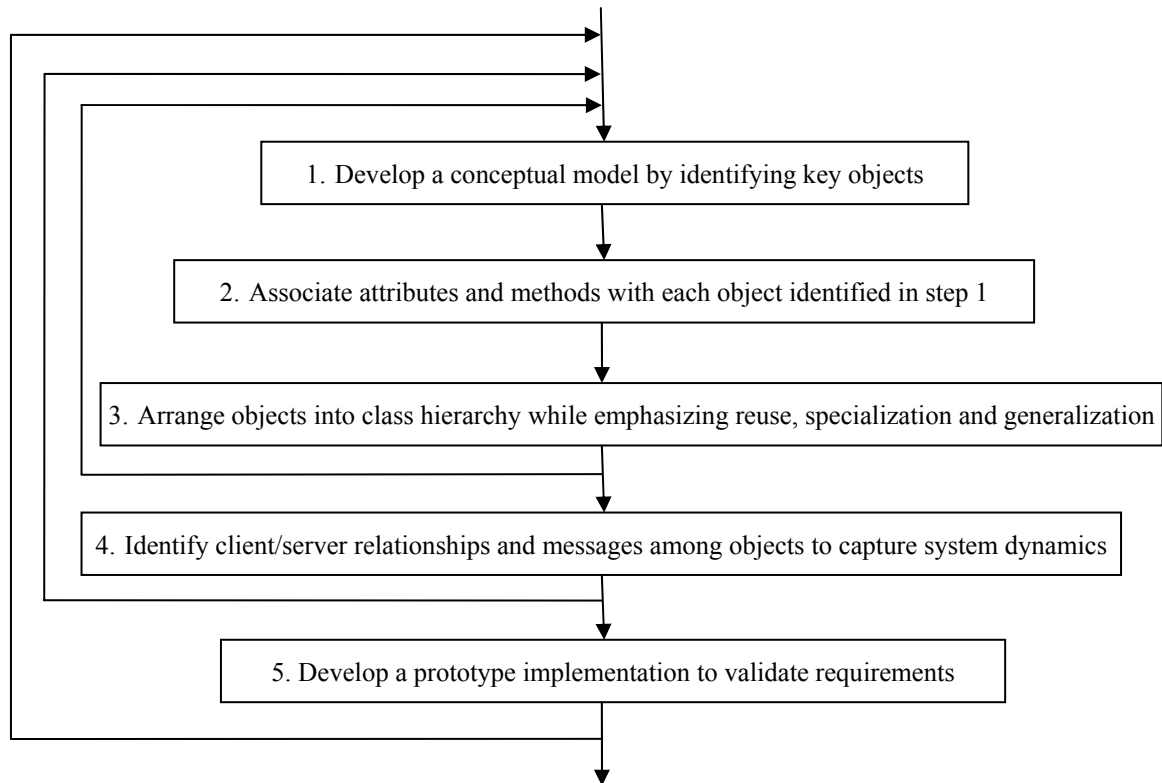


Fig. 2.1. Systematic procedure to develop a software system using the OO paradigm

From the viewpoint of functionality deployment, OO methods can be applied in four different modes to revolutionize the way product and process data is communicated and stored and the way applications are integrated (Conaway 1995): (1) in the development of the underlying database management (sub-)system; (2) as an application interfacing mechanism; (3) in the development of the application user interfaces; and (4) in the development of the body of middleware applications, which are common in the integrated architecture. Of these modes, widely recognized as one of the most important integration enablers is the second mode, which provides an OO software framework based on the distributed object technology to interface multiple applications. In this framework, the components of a system are normally defined as distributed objects and packaged as independent pieces of codes that can be accessed by remote clients by the method of invocation (Chen & Hsiao 1997). Typical distributed object technology includes CORBA (OMG 2002), Java Remote Method

Invocation (RMI) (Sun 2002) and Distributed Common Object Mode (DCOM) (Microsoft 1998). Good overviews of these technologies have been made by, *e.g.*, Urban *et al.* (2001) and Plasil & Stal (1998). Distributed object technologies in this sense are “lightweight”, robust and flexible for implementation to facilitate function-shipping and data-shipping in a distributed computing environment at any scale. Those PDM functions normally only embraced by large-scale companies with 20,000 to 50,000 users on the same system based on traditional distribution technologies can now be easily replicated in small and medium companies. This strongly justifies the novel two-step research efforts on system integration: firstly capture the desirable static and dynamic semantic relationships between all the CAX applications involved in a targeted engineering process; secondly, use OO methods to realize and maintain these relationships within the integration infrastructure without impairing the autonomy of each component application.

- Summary of the reviewed aspects driving system integration from design to manufacturing

The reviewed aspects affecting system integrations strongly justify research efforts having the following features. They should be undertaken accompanying comprehensive identification and characterization of a good application context where network-enabled integration strategy is indeed promising. Activity and data modeling are always required, and IDEF0 and EXPRESS or UML modeling methods respectively are recommended. Supporting CE strategy should be taken into account. Making the integration infrastructures as intelligent as possible is a must but the intelligence should preferably be compactly embedded within the corresponding software objects especially when no large-scale knowledge base is required. Data

integration and process integration are the most important integration functions to be incorporated in the system. OO methods should be used in the entire system development process to achieve an elegant, easy-to-understand system.

2.3. Review of Several Representative Integration Architectures

As stated, there is almost no widely accepted global reference architecture that can be easily geared to develop any specific integrated system across a set of distributed CAX applications. Researchers would rather develop their own architectures reflecting their own integration approaches on a project-by-project basis. This section examines seven representative architectures which define the components of a system and the relationships among those components.

- MICS and PTCS (Thomas & Fischer 1996)

The MICS architecture (Fig. 2.2 (a)) represents a hypothetical system that covers all major CAD/CAM functions and consists of four main components: a central database, a control module, application software packages with their wrappers and a communication channel. It does not support integration of distributed CAD/CAM software packages, although they can be plugged into the unified CAD/CAM system to realize data integration via the common database using wrappers and the communication channel. The control module is designed to organize and monitor the execution of activities in the system and automate the scheduling and execution of the activities. However, the MICS concept is only partially implemented with a subset of the overall CAD/CAM system in PTCS (Fig. 2.2(b)), which integrates two commercial CAD/CAM software packages, Pro-E[®] and ToolPro[™]. Further, there is only one wrapper for both CAD/CAM software packages and the wrapper is combined with the control module into a single module. The central database is connected to the

CAD/CAM system through the wrapper and control module. Both MICS and PTCS provide one consistent user interface to the CAD/CAM applications and thus a CE-compatible common computer environment for the different CAD/CAM functions. Comparatively, the PTCS system is a more tightly integrated environment which is near to a super-tool both having advantages and limitations from the viewpoint of integration.

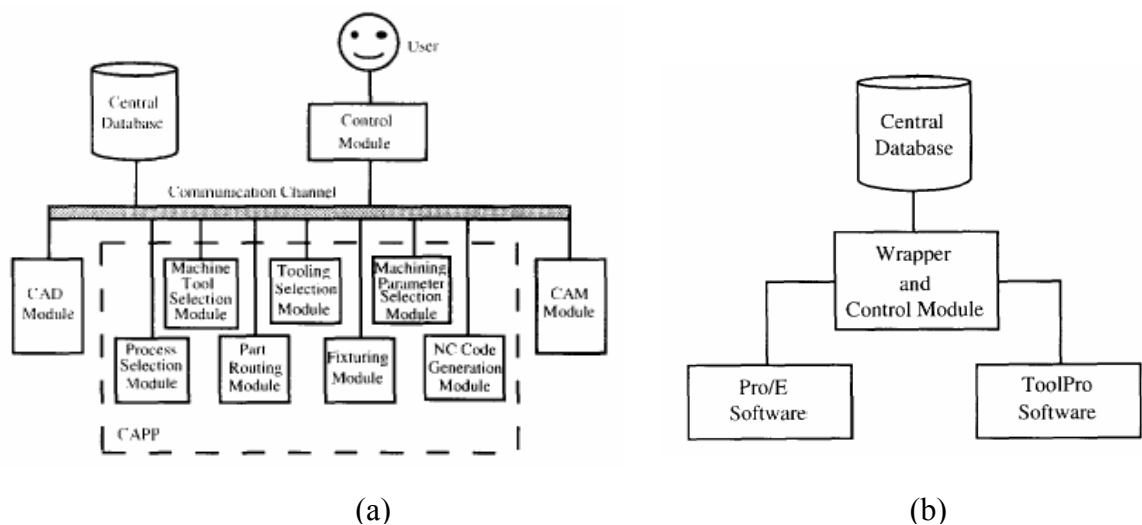


Fig. 2.2. (a) Modular integration of CAD/CAM software (MICS)
(b) The Pro/E-ToolPro CAD/CAM system (PTCS) (Thomas & Fischer 1996)

- SDM and IPDE (Urban *et al.*, 1996,1999a) (Fig. 2.3)

As mentioned above, the development efforts from SDM to IPDE mainly address data integration issues to facilitate effective sharing and management of the engineering data produced by a set of distributed CAD/CAM/CAE applications. It has a very clear definition of the architecture which includes the integrated product database (IPDB), the shared data manager (SDM), and a set of domain access interfaces (DAIs). It is not just another PDM system because it is a tightly integrated system with sufficient flexibility desired by its targeted application context. The DAIs provide interfaces that allow different applications to directly request services from the SDM. The versioning/configuration management mechanism and the database schema carry

semantics that are more specific to the application domain. The EXPRESS language is used in data modeling and the object-relational database Oracle[®] 8 is used to store the data. A key step is then taken to map the EXPRESS models into the Oracle[®] 8 database schema. Metadata queries and design data queries are performed and coordinated in the networked environment. Process management/integration is not covered because its application context is data-intensive with a simple process pattern imposed.

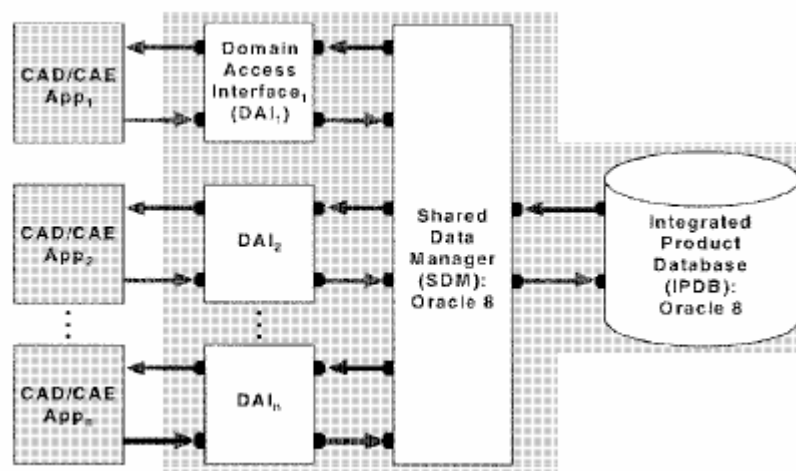


Fig. 2.3. The SDM and IPDE architecture (Urban *et al.*, 1996,1999a)

- A collaborative framework for concurrent net shape product and process development (Chen 1997) (Fig. 2.4)

This network-integrated engineering environment for development of net shape products and processes was designed as a client-server configuration. The servers include a process management server, a data server (managing material/tool specifics and standard components), a product data management server and a knowledge server. The client applications include four types of advisory tools, *i.e.*, the product design advisory tool, the process design advisory tool, the die or mold design advisory tool, and the die or mold manufacturing process planning advisory tool. Product and die/mould design tools are loosely integrated into the environment on the client side

through a workbench (workspace) and one or more bins (storage area) for the developer. Remote access and control for the client applications to the servers was enabled by PC Anywhere TM, which is very unfamiliar to the community and no further details about the underlying communication mechanism were given. Much effort can be found dealing with system analysis and modeling that makes it distinct from others by the presence of some clues on how the system functional modules are identified. The data modeling practice was not consistent all along but mixed with several approaches including the currently almost outdated E-R (Entity-Relationship) modeling approach. Although the integration functions seem to be comprehensive, it is unclear how the constituent servers and client applications are coordinated to work together and the process management function was not developed to the implementation level.

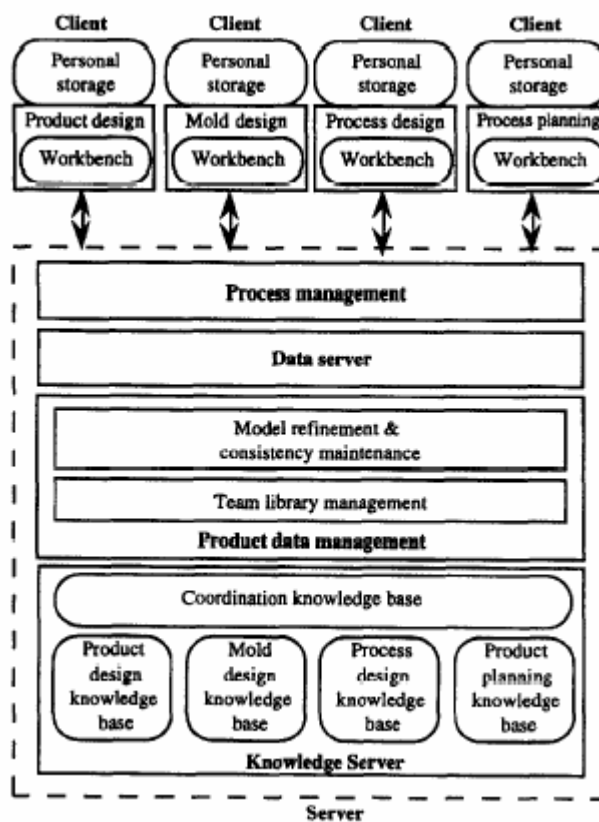


Fig.2.4. An integrated framework for net shape product and process development (Chen 1997)

- The CONCERT architecture (Hanneghan *et al.*, 1995, 1998) (Fig. 2.5)

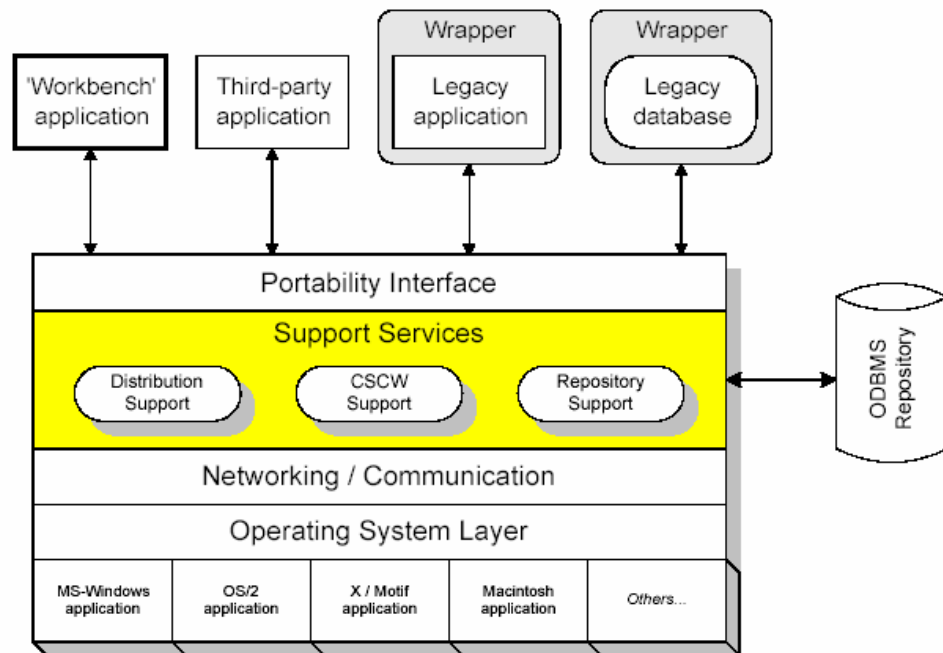


Fig. 2.5. The CONCERT environment (Hanneghan *et al.*, 1995, 1998)

The CONCERT (CONCurrent Engineering support) environment (Hanneghan, *et al.*, 1995, 1998) identifies three core support services that are considered important in distributed and integrated CE-compatible engineering environments. These highly co-operative components are the repository support service, the CSCW support service and the distribution support service. CAX applications to be integrated are viewed as third-party or legacy applications. The CONCERT environment provides a sample of highly-flexible integration architecture in which data integration, process integration and even CSCW functions can be incorporated. However, it is at so high a conceptual level with lack of developments towards implementation that it is unclear how even a simple specific function is finally realized at the implementation level.

- SUKITS (Westfechtel 2000) (Fig. 2.6)

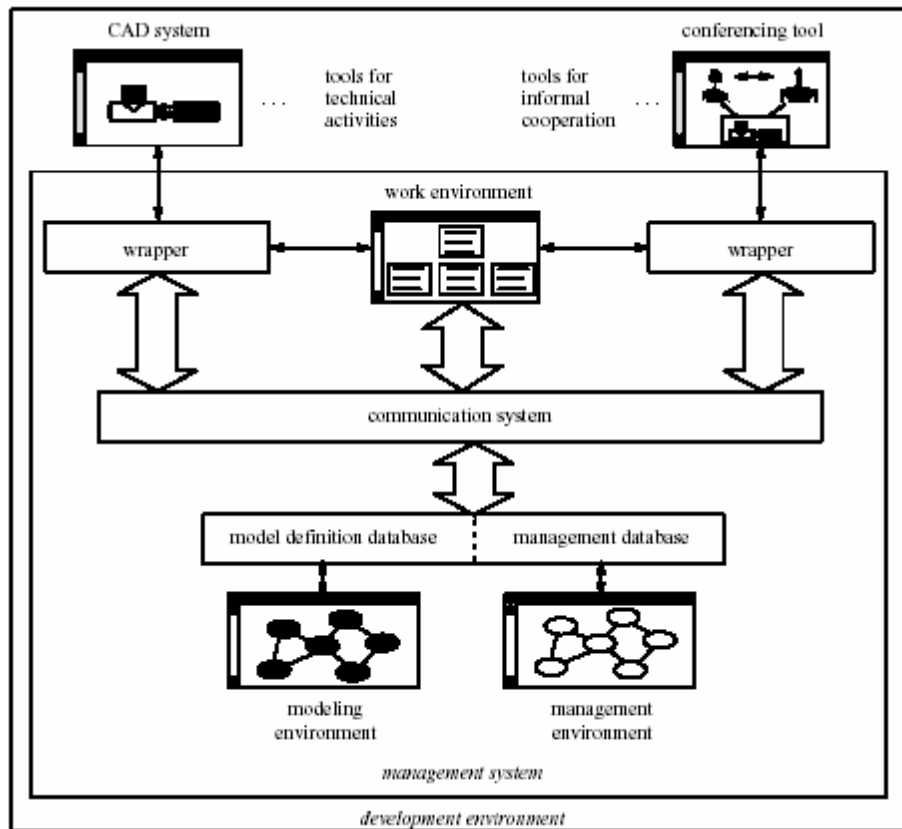


Fig. 2.6. Architecture of the SUKITS integrated development environment (Westfechtel 2000)

The data integration function, as well as process integration function in the SUKITS architecture, has been mentioned in the previous section. The entire architecture consists of a management system that is interfaced with a set of distributed tools for technical activities and informal cooperation. The management system is in turn composed of the following components: the communication system, the management database, the wrappers, the work environment GUI supporting tool activation and process management, the manager environment GUI supporting project managers and the modeling environment GUI to adapt the management system to a specific application domain. The SUKITS architecture is probably one of the most comprehensive distributed system integration from design to manufacturing. However, its conceptual framework is heavily based on the Graphic Theory and Graphic

Language-oriented (rather than OO) software and services originally devoted to a different application domain, software engineering. This makes its contributions appreciated by very few people. Further, some specific functions such as versioning control of interdependent documents are very restrictive because it overlooked many prominent distinctions between a mechanical engineering process and a software engineering process.

- The WWW-based integrated product development platform for sheet metal concurrent design and manufacturing (Xie *et al.*, 2001) (Fig. 2.7)

This proposed system seems to target at integrating all the design and manufacturing functions involved in the sheet metal product development process. It is claimed to consist of an unfolding module, a WWW-based data integration platform, design/manufacturing knowledge bases, data communication tools among different modules, a customer interface module, a RTCAPP module, a CAD module, a CAM module, a cost estimation and optimization module, a computer simulation platform and GUIs. However, it is not clear how these modules are logically organized into a unified system. It looks more like a description of a pool of non-integrated tools/modules although some of them provide some data integration functions shared by several others through using the WWW technologies and a web-based off-the-shelf PDM system, Pro/INTRALINK[®]. It is also unclear how the engineering activities performed by each design/manufacturing tool are integrated together to conclude a full development cycle by the end-users with the help of the integration modules. A lot of efforts were given to the product and process modeling (called information integration framework in the original paper) which tries to set up the desired interoperability from product design to downstream functions. However, only easy tasks were done there

and the whole description seems to be quite ambiguous from the viewpoint of relevance to the demonstration of an approach to develop an integrated system.

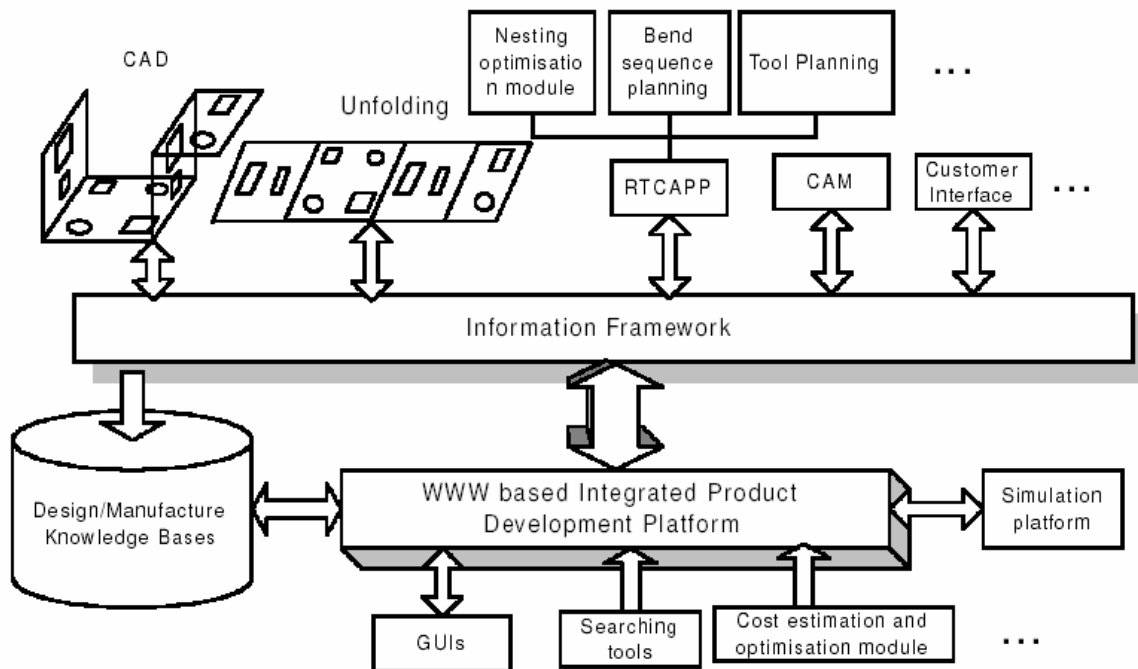


Fig. 2.7. The WWW-based integrated product development platform for sheet metal concurrent design and manufacturing (Xie *et al.*, 2001)

- The architecture for a CAD/CAPP/CAM integrated system (Wang & Zhang 2002) (Fig. 2.8)

This architecture supports integration of some distributed application subsystems including a feature-based CAD subsystem, a CAPP subsystem, a CAM subsystem, *etc.*, and some common tool service subsystems including constraint management subsystem, evaluation and decision supported subsystem, database management subsystem, *etc.* The main integration functions provided can be viewed as residing on the data integration level and heavy database schema normalization efforts can be noticed since the relational database system is used. No explicit process management functions are provided although some other alternative functions shared by two or more subsystems are included in the system. No distributed object technology is used

and the communication between subsystems is realized via low-level TCP/IP protocol on the Internet/Intranet network, which makes the developed system quite inflexible.

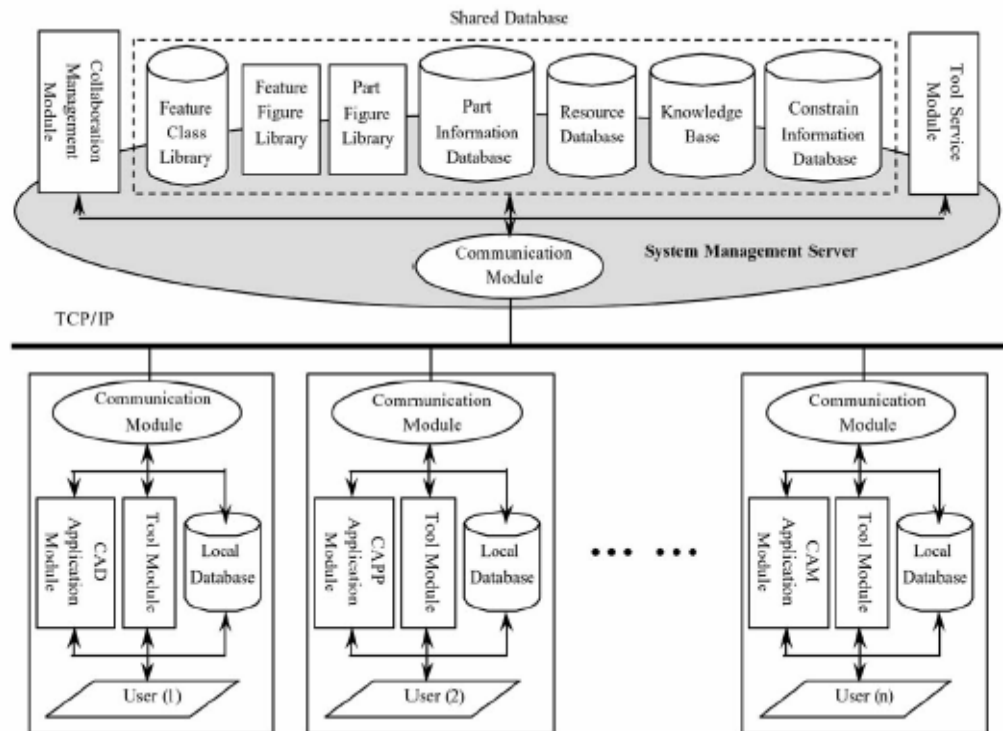


Fig. 2.8. Structure of feature-based collaborative development system (Wang & Zhang 2002)

- Summary of architectures

There is only one element, the data integration component, common to all of the integration architectures, discussed above. A data integration function thus seems to be the minimal ingredient of a design-to-manufacturing integration architecture. Process integration function is the second most likely to be included in the architecture. Many other cross-disciplinary functions, such as the CSCW services and large-scale knowledge base-enabled DFM/DFA (Design For Manufacture / Design For Assembly) services, may be incorporated into the integration infrastructure depending on the application context. One common rule is that the integration infrastructure only concerns issues such as user-interface unification, interconnections between participating CAX applications and providing common services shared by at least two

other modules. All of these architectures have some features justified in the previous section, but none of them have all the features summarized. Additional limitations of these architectures have been presented in Chapter 1.

A network-integrated engineering environment is always a large complex system. Architectures not only provide models representing different aspects of systems, but also provide approaches for integration functions. As the compendium to describe the interrelationships among components and the stages of system evolutionary trajectories, architectures are widely used to unify various component modules at different spatial levels and temporal stages, presenting the system as a holistic whole. Recognizing the significance of architectures, researchers are continuously improving existing architectures and probing for new ones. This thesis will begin developing an integrating architecture from a perspective completely different from those presented above.

CHAPTER 3

CHARACTERIZING FEATURE-DRIVEN ENGINEERING PROCESS

In this chapter, a unique perspective to address the complex engineering process is presented. From this perspective, one can identify a set of feature-driven engineering processes among its superset of more generic model-driven or data-driven engineering processes. Characterizing the feature-driven engineering processes is thoroughly studied in this chapter and regarded as the starting point to develop mechanisms for implementing an integrated engineering environment which explicitly incorporates these characteristics. The purpose is to reveal how an engineering process is decomposed and how to represent this decomposition, to identify different kinds of dependency relationships existing in the process and their properties, and to develop an adequate design transaction model to specify the interactions between a CAX tool and the shared data store. A collection of considerations that have not been covered in the current literature is highlighted wherever relevant.

3.1. Hacking the Complex Engineering Process: the Feature-driven Way

Although the interest in computer supported product development environment has entered the Internet era, it still follows a so-called data model-driven approach (Borja *et al.*, 2001) to improve a physical engineering process. This approach argues that

computer-aided engineering systems should be based on information data models in order to properly document the intermediate or final design results as a common communication medium for designers, analysts, manufacturers, and other product development people. As is universally acknowledged, the basic way to address a complex engineering process is to decompose it into a set of sub-processes and then (re-)integrate them as a whole, or briefly, “divide and conquer”. The data model-driven pattern makes it possible to decompose a complex engineering process into manageable sub-processes, each of which corresponds to a task-specific tool with a private database to yield a permanent data model. The model then becomes the information medium to enable process reintegration. However, due to the complexity, there are still several challenges when dealing with a practical process.

Firstly, no commonly agreeable criterion is available for the definition of the decomposition because it is completely problem-specific. A good practice always implies a good decomposition scheme, and the achievement of such a scheme requires in-depth understanding of the focused process towards adequate process re-engineering. Secondly, the constituent sub-processes defined in whatever way are semantically interdependent and the dependency relationships are always present implicitly at various information abstraction levels. This makes it difficult to be captured and enforced completely and precisely. Therefore, the fuzzy process knowledge is always only exploited by human users who are fully responsible for process control while using traditional engineering systems following the plain data model-driven approach.

Loosely speaking, downstream models are derived from upstream ones until a root model, *i.e.*, the product design model. In another words, some information elements in

one data model may be recursively incorporated into other models in a transformed or even untransformed form as long as they are interoperable. Yet the traditional engineering tools are unaware of the derivation relationships and treat the data models independent of those from which they are derived. The overlapping information has to be reentered manually. The information derivation process has to fully rely on the users and the derived models have to be generated from scratch even though it can be potentially generated in a fully or partially automatic way. Furthermore, the maintenance of the interdependence relationship is also dependent on the users to ensure that the desired data consistency always holds in case of change propagations among the interdependent models involved in a project.

To overcome these limitations of the normal model-driven approach, the feature technology has been employed to bridge the missing link between pairs of interdependent models. This is a promising mechanism to improve system integration and design automation. Based on this mechanism, relevant information, in terms of feature sets, encapsulated in one application model can be automatically derived from that in others through a so-called feature mapping process. Various feature mapping algorithms have been reported in different application contexts. They are all based on the same underlying principle: generating a new set of feature instances B from the given one A through knowledge-based reasoning supported by feature mapping knowledge bases (Zimmermann *et al.*, 2002). For example, in the progressive die design process, the flat pattern features can be derived from the corresponding sheet metal product features, the die operation features from the flat pattern features, *etc.* This characteristic has led to the development of an intelligent design automation

system to automate certain design steps, like the IPD (Intelligent Progressive Dies) system (Jiang *et al.*, 2000).

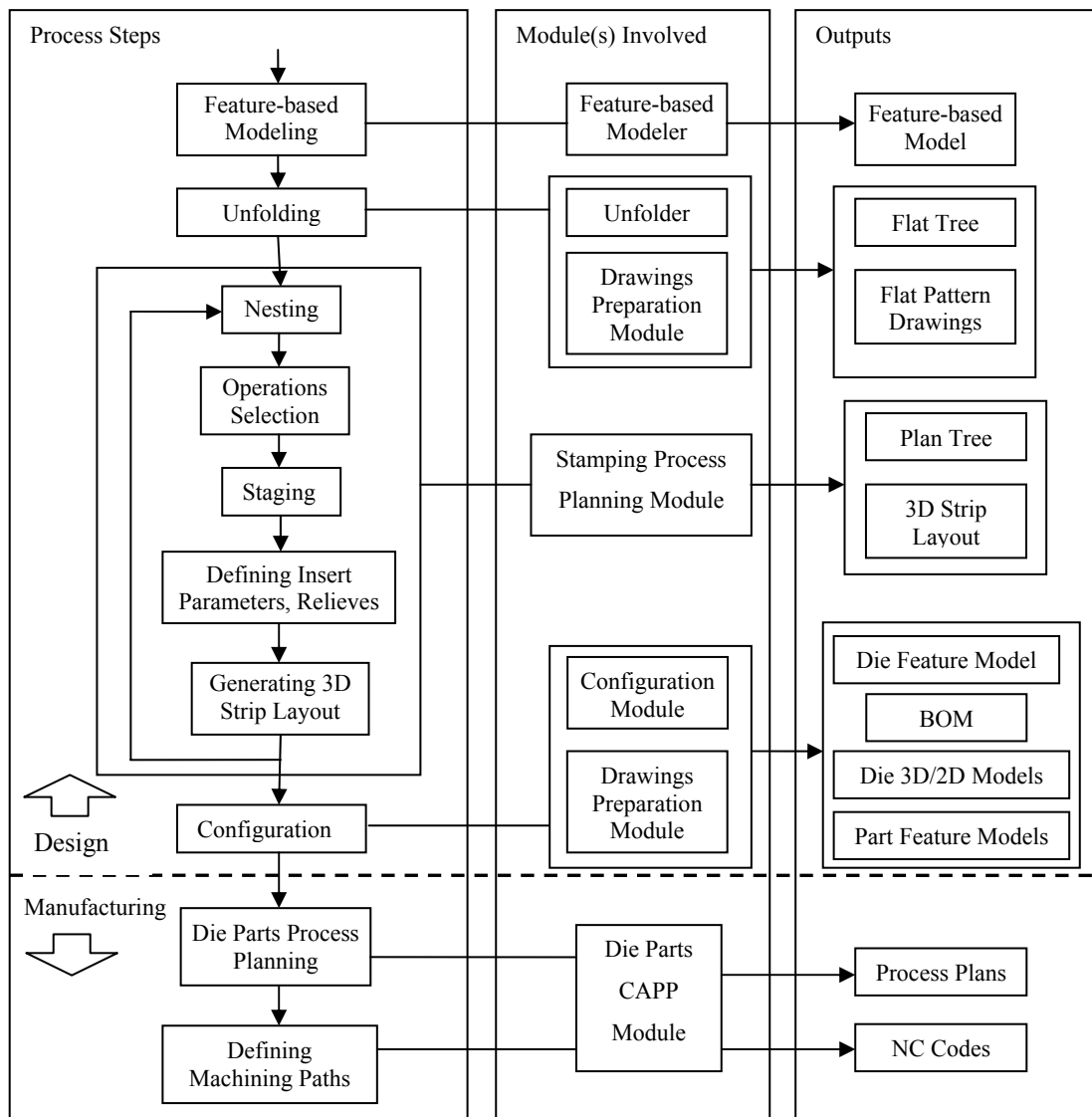


Fig. 3.1. The feature-driven progressive die design and manufacturing process

Once the model-driven process is further narrowed down to a feature-driven process, it becomes possible to characterize the dependence relationships with additional insights because the feature notion can help precisely locate the specific constraints reflecting the dependency relationships. A feature-driven engineering process is thus a special type of model-driven process. The specialty lies in that there is a clear feature-driven “track” in the feature-driven process because the missing links between all the models

involved are explicitly bridged by features. For example, if the progressive die design and manufacturing process is only roughly viewed as a model-driven one, all the outputs from each process steps are either ordinary geometrical models, or design data sheets, or NC code files. It is implicit how these outputs are related to each other, and the whole process is difficult to control. If the process is revamped as a feature-driven one (Fig. 3.1), on the other hand, the backbone of the information flow of the process is a structured net of feature-based models which clearly indicates how the current process step relates to others at various levels of information details (see also Fig. 3.7). It is then possible to more properly control the process run.

It is important to note that the definition of a feature-driven engineering process does not necessarily only include feature-based models in each step-process. It is not equal to a multiple feature mapping (or feature transformation, feature conversion) process since feature mapping restrictively refers to derivation of a task-specific feature model from other feature models (Shah 1988). The real central condition is that every step-process involves the handling of a feature-based model, either feature model creation, feature model mapping, or transformation of feature-based models into ordinary geometrical models, design data sheets and/or NC codes. Despite the promise of the feature technology for improving the understanding of the nature of an engineering process, extra efforts are needed because currently reported works related to using computer technologies to improve feature-driven engineering processes are only limited to the identification of mechanisms for feature modeling and automatic feature mapping. This is crucial in automatic or semi-automatic generation of individual feature-based models but has little relevance to the improvement of system level integration. As such, when one feature-based application in one domain is required to

co-work with systems and programs in other domains, the integration functions, such as information sharing and exchange, have to solely depend on the generic integration mechanisms without any unique augmentation based on its own inherent characteristics. It is therefore desirable to examine the nature of the feature-driven engineering process from the viewpoint of system integration.

As far as integration functions are concerned, it is found that a feature-driven engineering process can be further characterized in several new aspects with the target of developing a more relevant integrated engineering environment. Firstly, one can use the feature-driven “track” to identify all the model-model relationships to describe which model is dependent on other models revealing the global consistency requirements during the execution of the process. A feature-driven process can then be correctly configured to make the design state intuitively perceivable and manageable. This is called process management in this thesis. More details can be found in Chapter 6. Secondly, one can embed the interdependence semantics into the product data manager to provide enhanced version control and configuration management support. Specifically, the interdependence semantics is represented by a design object derivation graph consisting of a special kind of “*is-derived-from*” references. This “*is-derived-from*” reference does not exist between the new and the old versions of the same design object but between two different design object versions belonging to the same configuration version. Further, the ordinary “*is-part-of*” references found in common configuration manager have little relevance to the data consistency problem and thus are not a main concern. More significantly, the design changes due to version manipulations are identified to propagate in a special manner. A theoretical framework will be set up to reveal this special manner in the following. This framework expands

the taxonomy regarding the transformations between two feature spaces (feature_sets VS. feature_sets relationships) based on the feature space concept (Shah 1988) towards a definition of a new taxonomy regarding model-model dependency relationships between two design objects. Beginning from this interdependence semantics and the design change propagation property, a new version control and configuration management mechanism has been developed, which is elaborated in Chapter 5.

3.2. Process Decomposition and Information Flow

In order to successfully implement the overall integration of a complex engineering process, it is firstly required to properly subdivide it into sub-processes and devise the corresponding data models (Yoon & Shaikh 2000). Accordingly, one of the important procedures towards comprehensively characterizing a feature-driven engineering process is to develop an adequate process model at a high abstraction level to reflect the process decomposition and information flow semantics. The typical way to reach this goal is through process analysis and re-engineering. Upon process analysis, the relevant domain knowledge is extensively exploited, so that the functional decomposition of the targeted process under the “conventional” but most approximate to the ideal circumstance is comprehensively analyzed and formally documented as a benchmark for process re-engineering. Upon process re-engineering, the tasks that need to be performed includes evaluation of each sub-process for its contribution to the entire process, as well as redefinition of the contents of certain sub-processes and corresponding data models and/or adjustment of sub-processes sequence. One of the essential techniques for process analysis and re-engineering is the IDEF0 activity modeling technology (U.S. Air Force 1981), which provides a formal way to describe the relevant results in terms of a set of incrementally refined IDEF0 diagrams.

Throughout this thesis, the progressive die design and manufacturing process is taken as an example to demonstrate the core issues addressed in the course of developing a network-integrated feature-driven engineering environment. Consequently, the process is also exemplarily used in this section to show how a feature-driven engineering process is decomposed into manageable sub-processes. This inevitably makes some concerns probably only important for the exemplified process but not necessarily for other feature-driven engineering processes. However, the underlying approach is generic for all others.

Specifically, since a lot of literature has dealt with the progressive die design and manufacturing process (Cheok & Nee 1998a, b; Jiang *et al.*, 2000), little effort is required to perform process analysis and re-engineering. The task that is required to be addressed seems to synthetically use the dispersed knowledge to generate a formal process decomposition description in terms of a set of IDEF0 diagrams, as presented in the second part of this section. However, a closer look at those amounts of process knowledge revealed that one important process decomposition adjustment that has historically been made when a widely accepted intelligent and integrated design tool was introduced. Such an adjustment may now be viewed as a common sense, but the explanations currently available seem to be shallow and vague. Therefore, this section begins from a discussion of the said adjustment, which is viewed as a part of the effort for process analysis and re-engineering.

3.2.1. Moving Some Design Tasks in One Sub-Process ahead to Enter Its Upstream Sub-Process

The special process decomposition adjustment mentioned above occurs between two sub-processes within the progressive die design and manufacturing process, namely, die operation planning and die design. Conventionally, the former process includes nesting, operations selection and staging with the output being the *Strip Layout Description*; the latter process includes the design of punches, plates and the various types of ancillary components, with the output being the BOM and a collection of incrementally generated engineering descriptions for punches, punch plates, die blocks, *etc.* as well as all levels of assemblies (Fig 3.2(a)). By nature, iteration and feedback will definitely occur within each stage and even between stages. For example, a likely finding of insufficient consideration of the space requirement to place the punch on the punch plate when performing the die design may lead to re-staging of the stamping operations.

With the introduction of intelligent die design tools, many of the die configuration tasks can be automated provided that some primitive information elements are input manually at first. The automatic die configuration is done through rule-based and/or model-based reasoning which extensively exploits the configuration knowledge reflecting the built-in spatial and topological relationships between the constituent components. The natural way to specifying such a system only involves reorganization of the internal task contents within the die configuration sub-process. Concretely, those tasks included in the *Die Design* box need to be regrouped into two sets. The first set contains all the interactive operations to collect all relevant primitive information elements. The second set contains those vital die configuration operations automatically accomplished by the intelligent tools to generate all the electronic die configuration descriptions. Since such task reorganization only occurs locally without

relevance to the global process definition, the process decomposition logic is still identical to the conventional one (Fig. 3.2(b)). The reason why the first set of design tasks are collectively called *Interactive Design of “Insert Groups”* is given in the following paragraph.

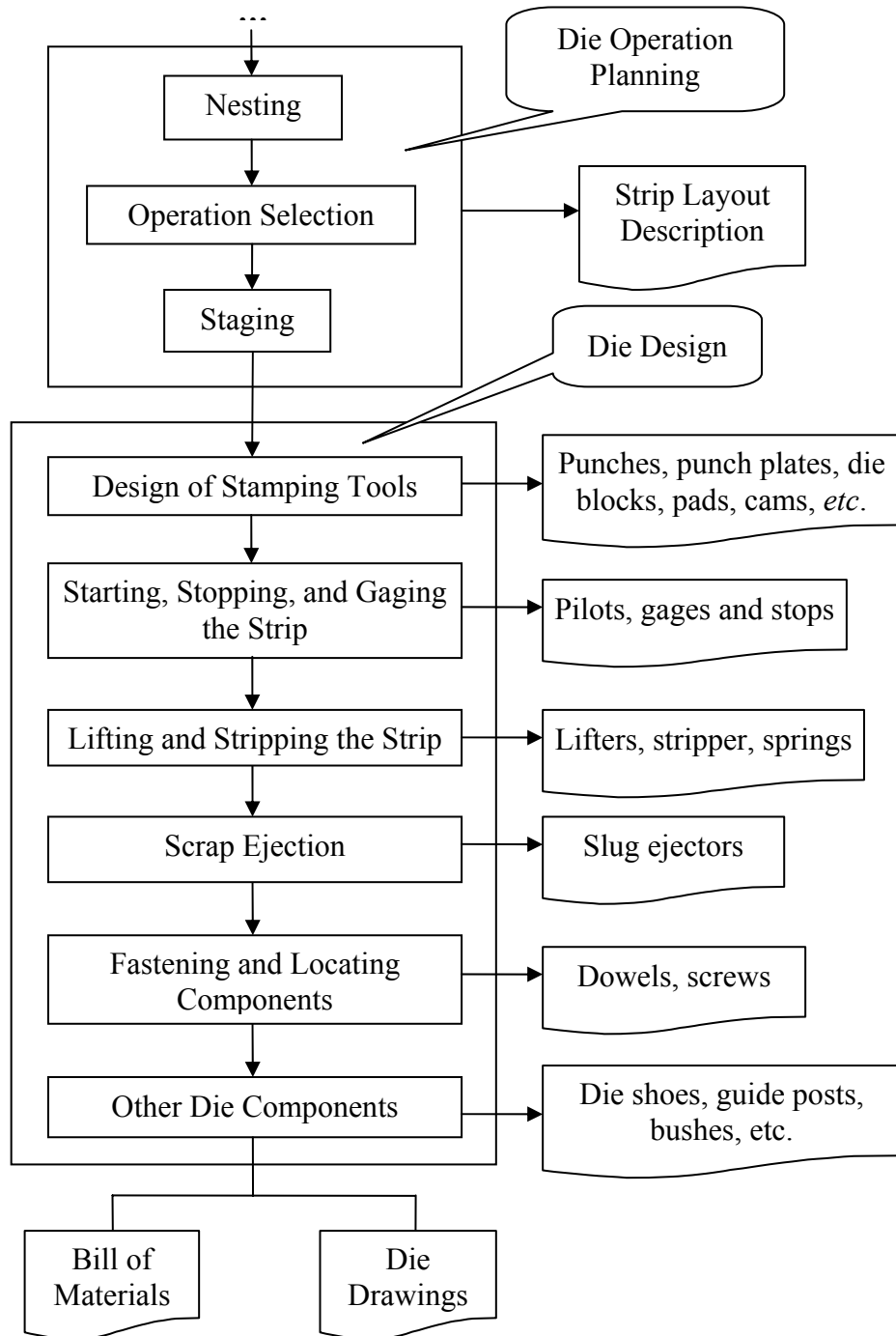


Fig. 3.2. (a) Process decomposition under normal circumstance (Nee & Cheok 2001)

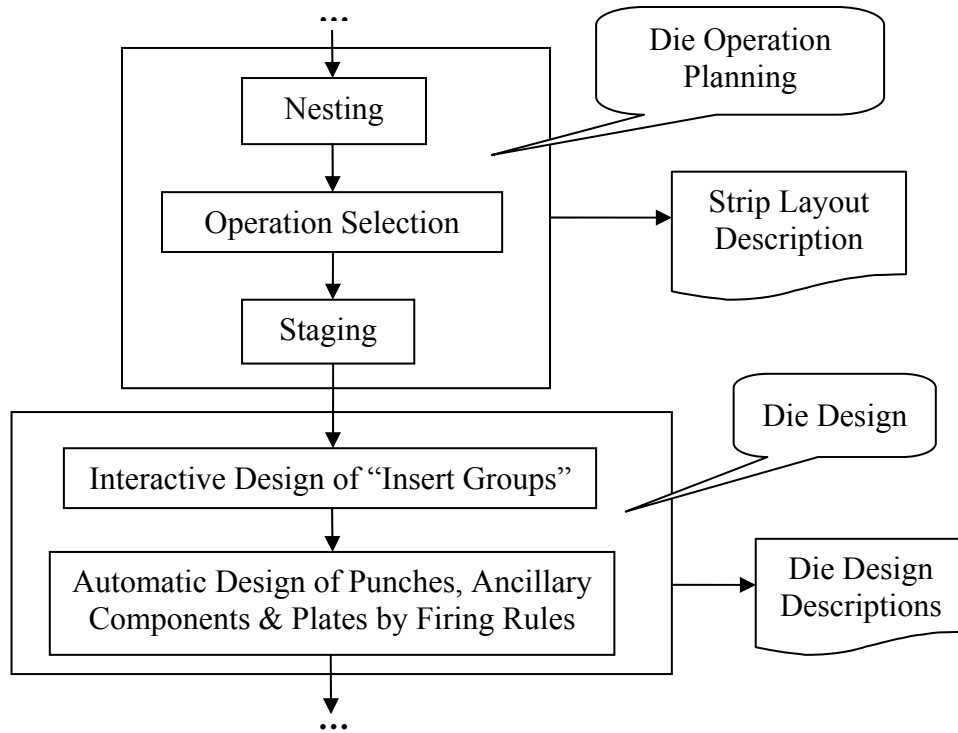


Fig. 3.2. (b) Process decomposition following the conventional logic

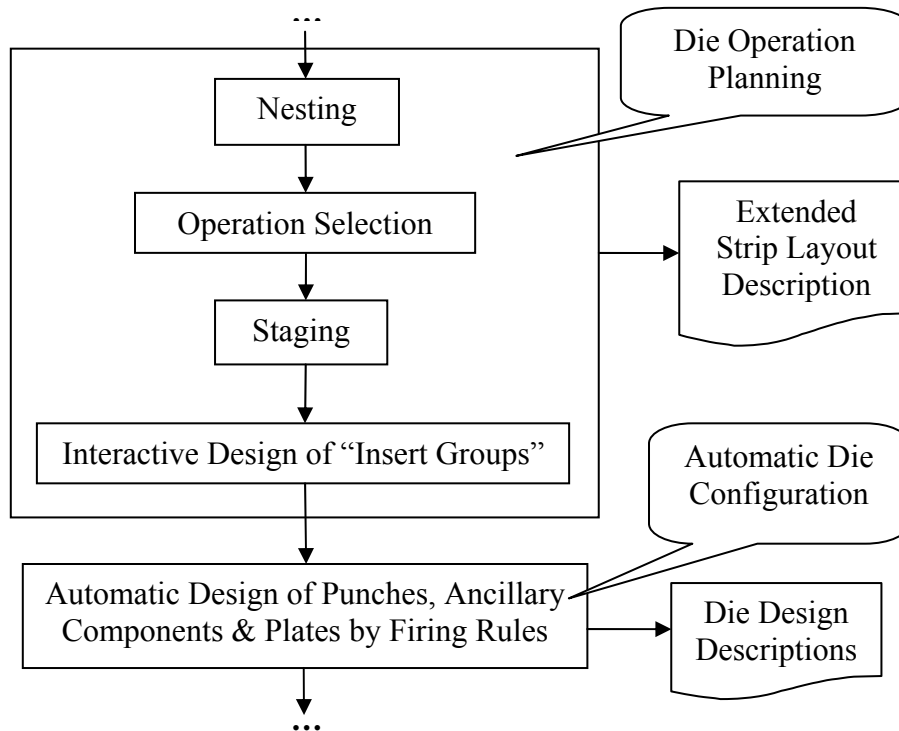


Fig. 3.2. (c) Process decomposition after process re-engineering

Extra considerations, once made, strongly justify a process re-engineering effort to move the first set of tasks within the die design sub-process ahead to become a part of

the die operation planning sub-process. To understand this, it is probably appropriate to begin from studying the requirements to support the execution of the first set of tasks. It is found that an extra data model needs to be devised to accommodate the ongoing interactive inputs which may last a long period of time across several turns of tool-runs. Further, the information elements in this data model can be partitioned into a number of clusters. Most of these clusters can be attached to the corresponding die operation descriptions contained in the strip layout model generated in the die operation planning sub-process. Although they may have no physical corresponding die operation description to attach, all other clusters can share a common information structure (class definition) with the above categories of clusters and a virtual corresponding die operation description may be applied. Therefore, if the first set of tasks within the die design sub-process are moved ahead to become a part of the die operation planning sub-process, the data models for each sub-process can be specified more rationally. Since a cluster of the information elements with the corresponding task to be moved can be loosely termed “Insert Group” or simply “Insert” (Jiang *et al.*, 2004), so the collection of this category of tasks is called *Interactive Design of “Insert Groups”*. Such a task move has some extra advantages. For instance, the end-user’s design operations can be more comfortable because the interactive operations are combined together while they are all actually performed in the same operation environment (*e.g.*, AutoCAD environment for IPD). Further, iteration and feedback can be conducted within one sub-process, which improves convenience and efficiency. Fig. 3.2(c) shows the alteration of process decomposition after performing the above process re-engineering. The die design sub-process is now becoming a pure automatic die configuration process with very limited interactions with the end-user. Each *Insert*

Group is now owned by a particular die operation and should be moved and updated accordingly when the operation is moved and modified.

3.2.2. Formulated Process Decomposition and Information flow: a Comprehensive IDEF0 Activity Model

Based on the process analysis and re-engineering results, a formulated process decomposition and information flow model can be constructed in the form of an IDEF0 activity model. Although the IDEF0 activity model is not implementable, it unambiguously captures the process knowledge and clearly sets the context in which data requirements and data flow for a system under development are defined. It also lays the foundation to provide a global view of the interdependence semantics in a feature-driven process (see section 3.3.1). A comprehensive IDEF0 activity model, including four IDEF0 activity diagrams named A-0 (Fig. 3.3), A0 (Fig. 3.4), A1 (Fig. 3.5) and A2 (Fig. 3.6) is developed for the progressive die design and manufacturing process in this thesis.

Diagram A-0 (Fig. 3.3) describes the activity A0 which performs the overall function of the system. It models the global context in which the progressive die design and manufacturing activity takes place. The activity has input data from a description model of the sheet metal product using progressive dies. Mechanisms of the activity are die configuration templates, standard components, machining resource descriptions, material stock descriptions, standard process models, machinability data and standard cost reference. The outputs of the activity are BOM, die drawings, cost estimate, process plans and NC codes.

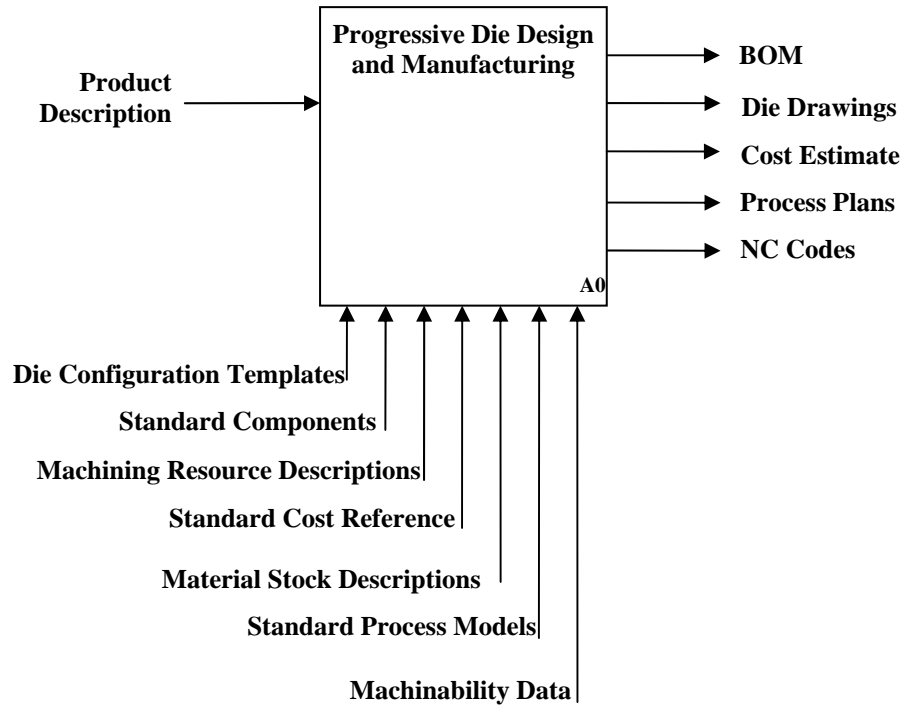


Fig. 3.3. Diagram A-0

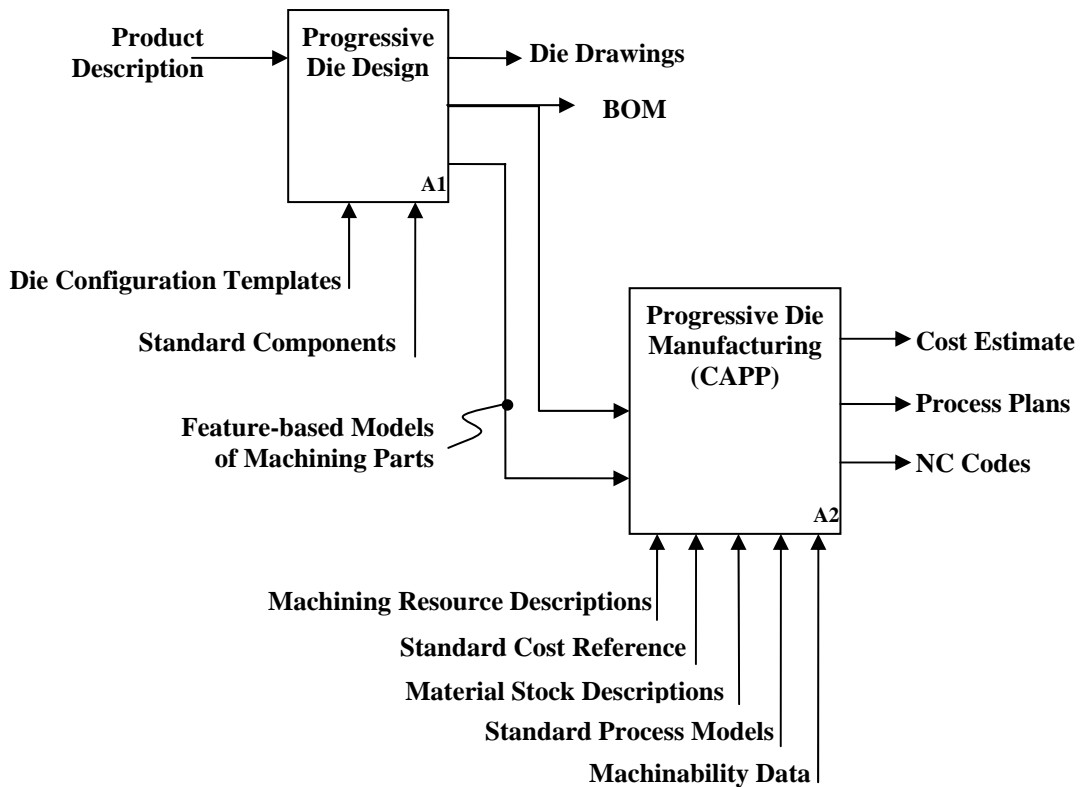


Fig. 3.4. Diagram A0

The overall activity on level A-0 is decomposed into two activities (A1 and A2) in Fig. 3.4, diagram A0. This diagram shows the relationship among the activities and the data inherited from the upper level (A-0).

Activity A1, shown in diagram A0 and expanded in diagram A1, performs progressive die design. Die drawings and BOM are generated in this activity. A group of intermediate feature-based models of the constituent machining parts are also generated. These models are then treated as input to the activity A2.

Activity A2, also shown in diagram A0 and expanded in diagram A2, performs progressive die manufacturing. It generates cost estimate, process plans and NC codes. Mechanisms of the activity are machining resource descriptions, material stock descriptions, standard process models, machinability data and standard cost reference.

Fig. 3.5 shows that activity A1 is decomposed into four activities. Activity A11 generates an electronic feature-based product model, which is used as input data by activity A12. Activity A12 generates a 2-D flat part drawing and an electronic feature-based flat part model, which is used as input data by activity A13. Activity A13 generates a 3-D strip layout model for users to check the stamping process planning result and an electronic feature-based die operation model, which is used as input data by activity A14. The 3-D strip layout model renders feedback information to the user and may be used as an input to activity A13. Activity A14 generates 3-D die models, die assembly tree (feature-based die configuration model), the BOM of the die assembly (including standard and non-standard subassemblies and components), all non-standard die component drawings and a range of electronic feature-based models

of all machining components in the die assembly. These feature-based models for the machining components are used as input by activity A2. The 3-D die models render feedback information to the user and may be used as an input to activity A14.

Fig. 3.6 shows that activity A2, which performs process planning for a machining part, is decomposed into four activities. Activity A21 generates the process sequence, which are used as input data by activity A22. Activity A22 generates process plans, which is used as input data by activities A23 and A24. Activity A23 generates NC codes, which are used as input data by activity A24. The feature-based model of the machining part is used as input for the first three activities A21, A22 and A23. Activity A24 validates the process plans and NC codes. It generates the cost estimates, validated process plans and NC codes.

In order to successfully implement the overall integration of a complex engineering process, it is required to properly subdivide it into proper sub-processes and devise the corresponding data models (Yoon & Shaikh 2000). Without doubt, such subdivision with corresponding data models will not have only one resolution. The activity IDEF0 activity models are used to document, compare and analyze multiple alternatives. Process re-engineering may be conducted in the course of searching the optimal resolution. The IDEF0 models presented above are exactly the final result based on a great deal of efforts of documentation, comparison, analysis and process re-engineering.

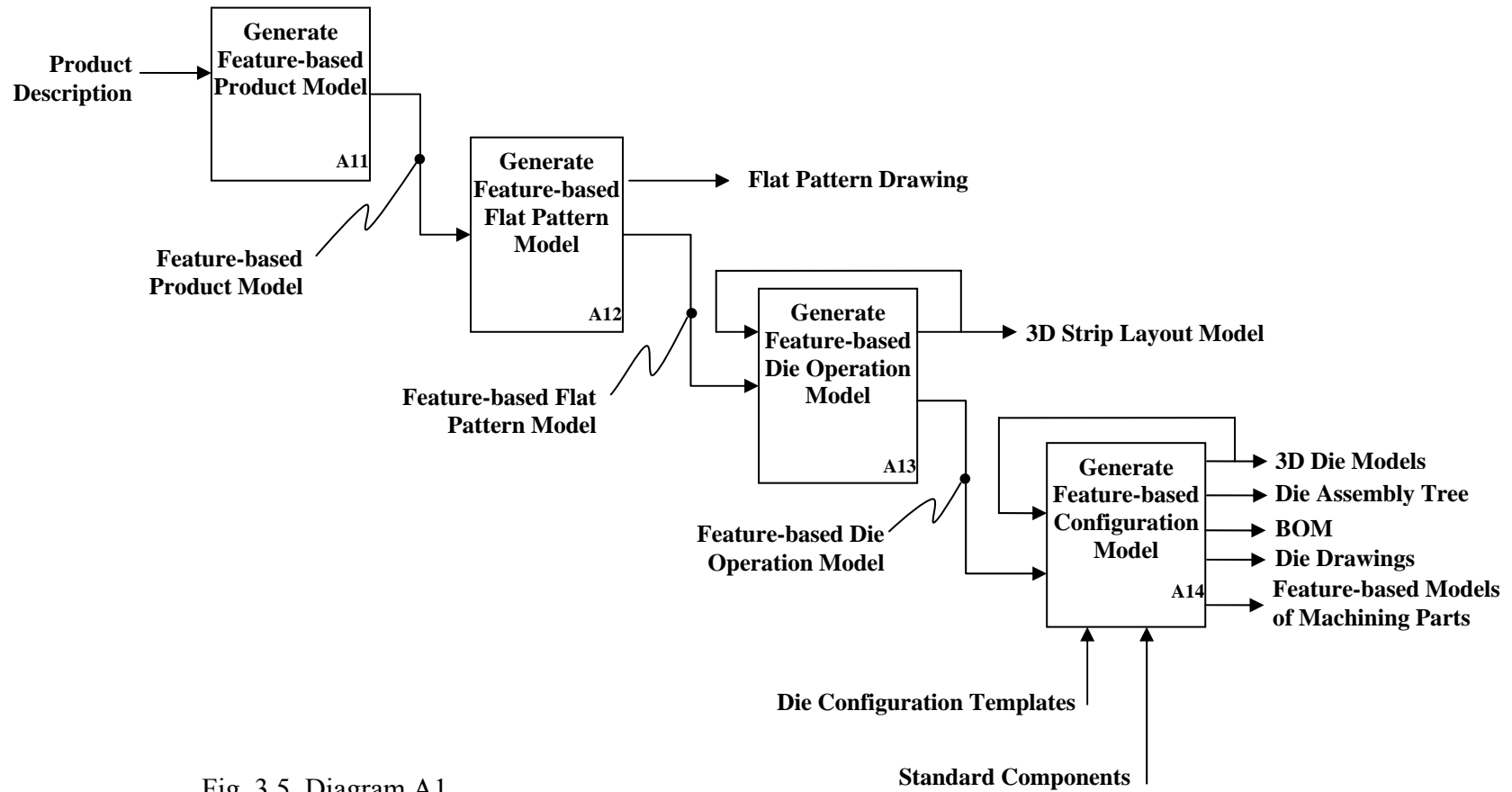


Fig. 3.5. Diagram A1

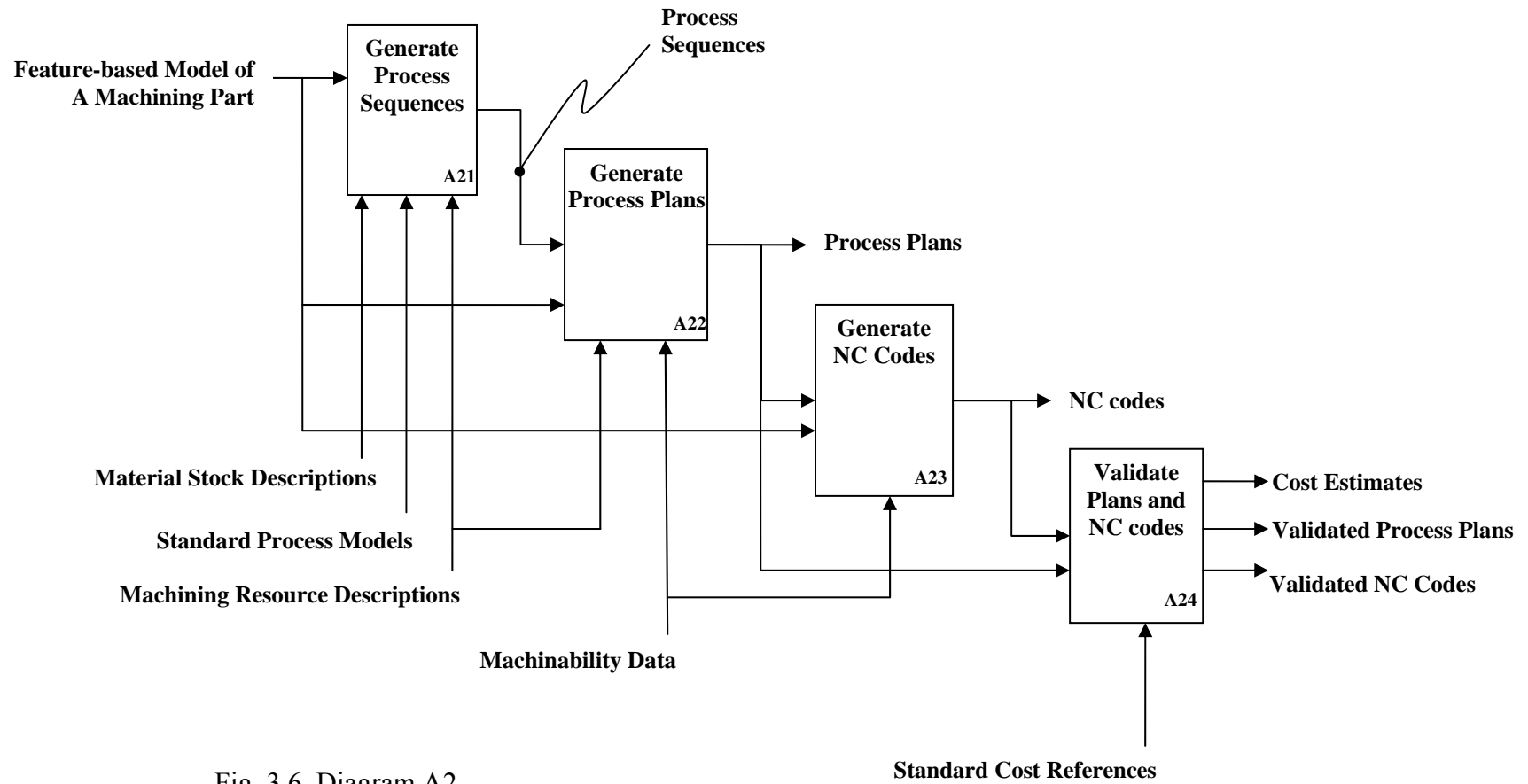


Fig. 3.6. Diagram A2

It should be noted that the description of the activities of an engineering process using an IDEF0 activity model can be recursively refined up to greater detail levels until the model is semantically descriptive enough to meet the requirement for decision-making or implementation. The model shown above only decomposes the progressive die design and manufacturing process into such a detailed level that every activity produces a complete intermediate feature-based model or final engineering renderings assisted by an individual CAD/CAM tool. In other words, creating internal information models (engineering databases) within corresponding computer-based tools participating in an integrated engineering environment is fully realized in one activity, but not necessarily across several activities. For example, the die operation model is fully created by the single activity A13, although A13 can be further decomposed into sub-activities, such as Selecting Piloting Method, Selecting Punches, Selecting Extrusion Operations, Staging, *etc.* (Cheok 1998). This is different from the ways adopted by some other IDEF0 models, in which the activities are decomposed into such a detailed level that one information model (document) generated in a computer-based system may be realized through several activities. For example, the conceptual design IDEF0 activity model developed by Feng & Song (2000) encompasses a group of activities, which collectively represents a whole sub-process generating one single information model (database). Typically, an IDEF0 activity model at a level more detailed than that of the current model is used to capture the database schema of the information model or the concrete design process knowledge provided for users to use a task-specific engineering tool. The current IDEF0 model is used to capture characteristics and requirements of the interoperability between the various design and manufacturing phases, each of which generates an intermediate model or final engineering renderings. How these intermediate models are achieved and represented

in the computer-based tools kernel is not directly relevant to developing an appropriate system integration infrastructure, which only concerns the global view of these models and tools. Therefore, the current model has been at an exact decomposition level, not too high while not too low.

3.3. Interdependence Semantics and Design Change Propagation Property

The interdependence semantics has been addressed in a few researches mainly as a secondary topic with respect to version control and configuration management. This section presents a formulated description on the interdependence semantics in a feature-driven process using the notions and terminologies found in the version management community.

3.3.1. Global View of Interdependence Semantics in a Feature-driven Process: Design Object Derivation Graph

As widely acknowledged, the ordinary versioning problem mainly concerns the management of the versioned complex and evolutionary design objects involved in designing artifacts consisting of components, which themselves in turn recursively consist of lower level components till the leaf level primitive components (Chou & Kim 1986; Ramakrishnan & Janaki 1996). In this case, when the users retrieve a version of a component, they need to be provided with the knowledge about its evolutionary track through a collection of its past versions and its lower level component versions used. In general, the component version evolutionary history is maintained by a version derivation graph (VDG). The VDG of a versioned object vo consists of a tuple $(\mathcal{V}, \mathcal{D})$, where \mathcal{V} is a non-empty set of versions of vo and \mathcal{D} is a set of directed edges in \mathcal{V} . Likewise, the *composite reference (is-part-of)* relationship is

maintained by a hierarchical diagram called a composition graph (CG) for the highest level object, the composite object, of an artifact in the hierarchy (Westfechtel 2000; Miles *et al.*, 2000; Park & Yoo 1995). A CG of a composite object co is defined as a tuple (CO, CR) where CO is a set of component objects and CR is a set of directed edges on CO .

For product data management of a feature-driven engineering process, the most important semantics which needs to be captured, is also a type of *derived-from* or *dependent-on* relationships existing in the collection of all versioned design objects involved in a process. However, this *derived-from* relationship is not between the new and old versions of the same versioned object but between two different objects probably labeled with the same version number to show that they share the same evolutionary pace. Similar to CG, this *derived-from* relationship can be represented by a design object derivation graph (DODG). A DODG for a specific feature-driven engineering process is defined as a tuple (DO, DR) where DO is a set of design objects and DR is a set of directed edges on DO . If (do_1, do_2) is in DR , do_2 is then said to have a derivation reference to do_1 , or do_2 is derived from do_1 . The main design objects in DO are feature-based models which are chained to act as the backbone of an engineering process initiated from an upstream engineering phase to the down-stream ones, including some concurrent ones in-between. From the feature-based models in the backbone, some ultimate engineering documents may be derived. Fig. 3.7 shows a DODG for the development process of sheet metal products using progressive dies based on the activity model presented in the previous section. A range of feature-based models, product feature model, flat pattern feature model, die operation feature model, die configuration feature model and a group of die part feature models constitute the

backbone of the entire engineering process. Flat pattern engineering drawing, 3-D strip layout model, die BOM and engineering drawings, die part process plans and NC codes are the ultimate engineering documents derived from the related feature-based models in the backbone. Typically, knowledge-based intelligent engineering tools are used for the derivation of the ultimate engineering documents from related the feature-based models and down-stream feature-based models from their upstream ones while the ultimate engineering documents may be further refined by the help of universal engineering tools, such as AutoCAD.

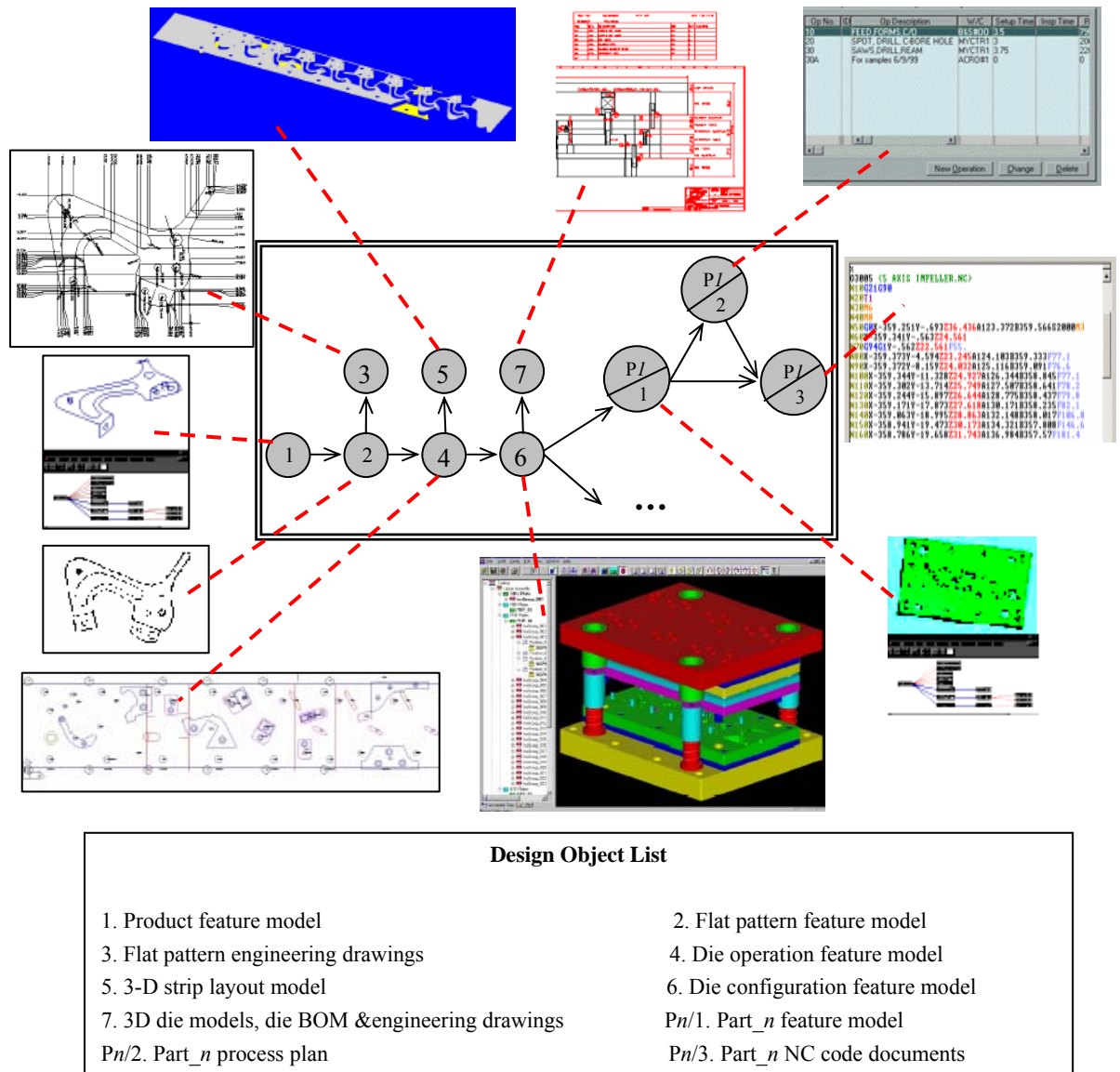


Fig. 3.7. The Design Object Derivation Graph

The DODG conceptually renders a full picture of the product data configuration of a feature-driven engineering project. It unambiguously specifies the information models to be generated and their dependency relationships. This makes all the product models, if mono-versioned, manageable by a certain dedicated integration infrastructure with assistances for users to track and control the data generation process mono-directionally from a root model towards the downstream tasks. However, a practical engineering process needs to handle multi-versioned design objects. Once a new version of a design object in between the process is created, the design change may propagate upwards and/or downwards, and thus cause the process to proceed bi-directionally, which generates new versions for both upward design objects (upward propagation versions) and downward design objects (downward propagation versions) in DODG (Fig. 3.8). The design change propagations through all the individual pairs of interdependent design objects are not identical, and it is necessary examine these with more details.

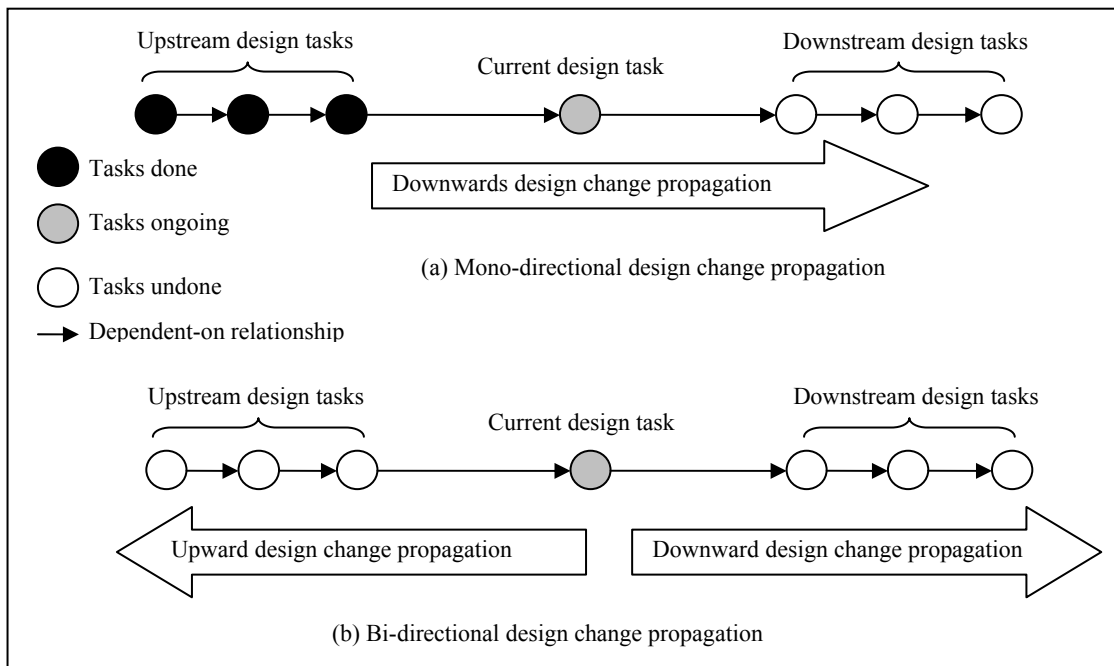


Fig. 3.8. Two types of design changes

3.3.2. Expanding the Feature Transformation Taxonomy Towards Dependency Relationship Taxonomy

With the global view of the dependence relationships in an entire engineering process being revealed, the subsequent sections will concentrate on characterizing the local model-model dependency relationships. In brief, they are classified into several categories, each of which has special design change propagation characteristics. Since the classification is an extension to the feature transformation taxonomy, which is based on the feature space concept introduced by Shah (1988), this section first examines the feature space concept and the feature transformation taxonomy.

According to the feature space concept, the totality of information related to a product, in all its aspects, over its entire life cycle, and for all conceivable applications, defines a domain called a *feature hyperspace*. The actual feature spaces for the life cycle applications of a given product are subsets of this hyperspace. Various types of relationships can exist between two subspaces, which may be of the same or different dimensions. Feature spaces of the same dimension may be partially overlapping or completely disjoint. In the overlapping regions, one can find features with identical semantics. Between the feature spaces of unequal dimensions, information from a higher-dimensional domain may be selectively abstracted to suit a lower-dimensional domain. This is referred to as a projection transformation from n to $(n-m)$ space. Another possible feature space relationship is referred to as conjugate spaces, which contain features composed of different variations of the same elements. The adjoint space is another relationship created by associating elements in one subspace to certain elements in another subspace. Accordingly, four types of feature transformations between two feature sub-spaces exist, namely, identity, projection, conjugate and

adjoint transformations. For information transfer from one domain to another, there are also distinctions between unitary and multiple transformations.

Apparently, the taxonomy given above is not a complete one, and like many other research works on feature mapping (Bronsvort & Jansen 1993; Shah 1988; Wong & Leung 1995, 2000), the classification criterion used was vague: it interchangeably measures the relationships between the entire feature models (feature spaces) or that between specific partial feature sets encapsulated in the feature models. Based on these observations, a fairly complete taxonomy is developed and presented next. The classification criterion measures the relationships between any two data models involved in the feature driven engineering process, including the non-feature-based models, which are the final engineering outputs derived from a feature-based model in the backbone of the DODG.

3.3.3. Model Derivation Function

Expanded from the abstract concept of feature space, mathematical functions are used to describe the possible relationships between interdependent models. At the highest level, when one model M_B is derived from M_A , the model derivation function can be loosely denoted as

$$M_B = f(M_A) \tag{3.1}$$

The information elements in M_B and M_A are not limited to features, but can be of any types that a feature-driven engineering process may involve. This function covers the case where $d_{M_B}/d_{M_A} = 0$, which means M_B is fully independent of M_A .

The closest relationship between M_B and M_A is probably that the information elements in M_B are variations of that in M_A (conjugate transformation), which is denoted as

$$M_B = f_c(M_A) \quad (3.2)$$

In most cases, M_B may contain information elements with no correspondence in M_A and vice versa. Let ΔAB denotes the information elements in A with no correspondence in B , and ΔBA denotes information elements in B with no correspondence in A . Another imaginable relationship will be information filtration (project transformation), which is denoted as

$$M_B = f_c(M_A - \Delta AB) \quad (3.3)$$

Likewise, the information addition (adjoint transformation) relationship is denoted as

$$M_B = f_c(M_A) + \Delta BA \quad (3.4)$$

Unifying equations (3.2), (3.3) and (3.4), a generic derivation relationship between M_B and M_A can be integrally represented by

$$M_B = f_c(M_A - \Delta AB) + \Delta BA \quad (3.5)$$

from which equation (3.2), (3.3) or (3.4) can be seen as one of its special cases, *e.g.*, when $\Delta BA = \Delta AB = \Phi$, it becomes (3.2).

While ΔBA denotes information elements in B additional to A , it can be further decomposed into two parts: those that can be fully deduced through a function f_a with some elements in A as the arguments and those which are newly added independent elements, *i.e.*,

$$\Delta BA = f_a (M_A - \Delta'AB) + \Delta'BA \quad (3.6)$$

According to its “greyness” (the extent of awareness), the conjugate function f_c and the addition function f_a can be classified into three types, “black box”, “white box” and “grey box” transformations. In the case of "black box" transformations, the data sources and targets within a model pair are related through a transformation and to each other at a coarse-grain level. It can be determined that the data sources and targets are related through the transformation, but no data target can be precisely expressed as a specific function of a (set of) data source(s). In the case of "white box" transformations, the data sources and targets within the model pair are related through a transformation and to each other at a fine-grain level. Every data target can be precisely expressed as a specific function of a (set of) data source(s). In the case of “grey box” transformations, data sources and targets within the model pair are related to a transformation and to each other at a medium-grain level. Only a portion of the data targets can be precisely expressed as a specific function of a (set of) data source(s). In this sense, adding new information elements, either in the process of creating a new model or deriving a new model from a source model belonging to an upstream domain, can always be seen as a “black box” transformation where data sources are Φ . Consequently, at the collection level, the model derivation relationships from model A to B represented by equations (3.5) and (3.6) can also be classified into three types. If $(\Delta BA = \Phi) \cap (f_c \text{ and } f_a \text{ are both “white box”})$, the model transformations are “white box”; else if $(f_c \text{ and } f_a \text{ are both “black box”})$, the model transformations are “black box”; or else, the model transformations are “grey box”.

Classifying feature model transformations into three types illustrates the fact there exist three ways to generate a target model from a given model or from the beginning.

For “white box” transformations, a fully automated design tool can be employed to realize the transformations merely through a push of a couple of buttons. For “black box” transformations, the knowledge about the derivation of a target model from a source model has to be kept in the designers’ mind and the target model has to be generated from scratch manually through an interactive design tool like creating a completely new model. For “grey box” transformation, a semi-automated design tool can be employed to automate part of transformation operations while manually realizing others.

Using the feature-driven process shown in Figure 3.5 as an example, the process to derive the flat pattern model is a **conjugate**, “**white box**” and **fully automatic** transformation; the processes to derive the flat pattern engineering drawing, 3-D strip layout model, die BOM and engineering drawings, and die parts feature models are all **projections**, “**white box**” and **fully automatic** transformations; the processes to derive the die operation feature model, die configuration model, and die part process plans are all **mixed**, “**grey box**” and **semi-automatic** transformations. The derivation of die parts NC codes is a process to transform two source models into one. The derivation function is similar to equation (3.5) which has analogous properties and the process itself turns out to be a **projection**, “**white box**” and **automatic** transformation.

3.3.4. Design Change Propagation Property

This paragraph discusses the design change propagation property on how it lays the constraints for implementation of the data integration tools, specifically, the version control and configuration management tools. The details of the versioning control and configuration management concept are given in Chapter 5.

Existing research works dealing with design interdependency plainly assume that the interdependent models always affect each other and the design changes always propagate upwards and downwards (Westfechtel 2000; Baldwin & Chung 1995). However, when examining the interdependent relationships represented by equation (3.5) more closely, it can be found that the change of one model will not necessarily always cause a corresponding change to its immediate interdependent upstream models or downstream models. Specifically, when the changes are only limited to the part ΔBA (current model is M_B and upstream interdependent models are M_A), they will not propagate upwards. Likewise, when the changes are only limited to the part ΔAB (current model is M_A and downstream interdependent models are M_B), the design change will not propagate downwards. Therefore, some design changes to a model in the DODG may propagate throughout the whole DODG, while others may only affect their near neighbors or have no effects on any neighbors. Furthermore, the determination of the design change propagation scope requires the knowledge of the specific information sets that have been changed and a highly intelligent “inference engine” to make adequate decisions based on these information sets. Since the version control and configuration management tool does not concern the interior information contents of the design object, making these decisions should be fully up to the designer. The promise of the version control and configuration management tool is to provide a comfortable context to execute the design change propagation scope after the decision is made during the versioning process.

The difference between the automatic and manual (including semi-automatic) transformations has influences on the propagation property. For automatic propagation,

it will be meaningless to proactively perform design changes to the models derived from the upstream ones. The only way is to change those from which they are derived from. On the other hand, for manual propagation, the models can be changed to evolve to a new version either proactively or reactively.

Using the feature driven-process shown in Figure 3.7 as an example, the flat pattern engineering drawing, 3-D strip layout model, die BOM and engineering drawings, die parts feature models, *etc.*, are all models derived automatically and thus cannot directly and actively execute design changes while other models like the die configuration feature model allow for both proactive and reactive design changes. Furthermore, if a design change to the die configuration model is related to the shape of the punch, this design change is expected to propagate both upwards and/or downwards; if it is only related to a die plate, it can only lead to downwards design change propagation.

3.4. A Special Design Transaction Model for Feature-driven Engineering Process

The previous section has shown that there are three ways to transform input data model(s) to output data model(s), namely, “white-box”, “grey-box” and “black-box” transformations. Consequently, three types of tools, automatic, semi-automatic and manual (interactive) engineering tools exist corresponding to the data manipulation ways. Such a rough classification is probably insufficient for dealing with registering an engineering tool into an environment. This section goes a little further from this point to identify the ways by which an engineering tool manipulates relevant data housed in the environment through design sessions. The implication of the data manipulation means is then studied with the target to develop an adequate design

transaction model to formally describe how engineering tools access design objects in the shared repository.

3.4.1. The Means by Which an Engineering Tool Manipulates Relevant Data through Design Sessions

An engineer starts interacting with an environment by initiating a design session which may span minutes, hours or even days. A long session may be decomposed into a set of short design sessions connected by saving and reloading an intermediate data model temporarily stored in a data store maintained by the environment. When the designer ends the session, an implicit save/check-in operation is issued for all relevant objects. Corresponding to the three types of engineering tools, there are four possible means by which a tool manipulates relevant data through design sessions (Fig. 3.9).

The most popular data manipulation means is found with tools that heavily depend on user interactions and work like an editor (Fig. 3.9(a)). When a new design session begins, the engineering tool optionally loads the input design object(s) or reloads an intermediate design object into its working memory space and maps them into an incore (in-memory) data structure. The incore data structure can be distinct from the actual data format used for physical storage. When the design session pauses for some reasons, a SavePoint is created by the tool to make persistent the incore data structure in the environment. The final design result is a special SavePoint, which is no longer reloaded for revision by the tool. Therefore, the operation to create a SavePoint or the final design result is almost identical.

A variant to the above data manipulation means is shown in Fig. 3.9(b). The optional input design object(s) become(s) mandatory and must be reloaded again along with the intermediate data model maintained as the SavePoint when a new design session is resumed after a pause. The consumed design object(s) may be overwritten by other designers during the interval between the first and the current session starting point. This is a risk of damaging the data integrity. Fortunately, this risk can be removed by incorporating all the information within the consumed design object(s) into the incore data structure and further the corresponding output design object immediately after the first design session has been launched. In this sense, this case becomes the case shown in Fig. 3.9(a).

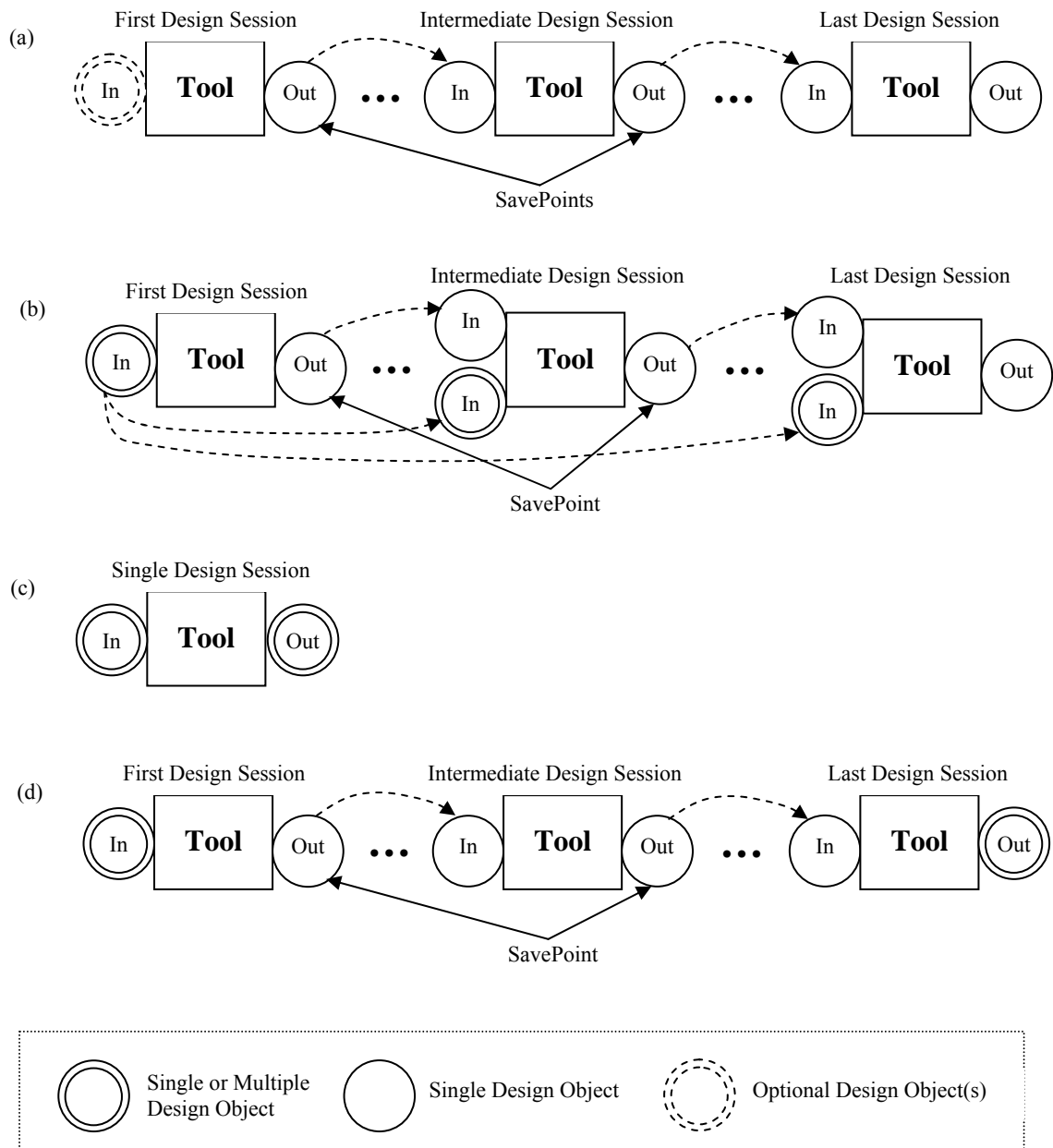


Fig. 3.9. Four possible means by which a tool manipulates relevant data

Another possible data manipulation means is found with automatic engineering tools. There is a single design session which automatically transforms the consumed design object(s) into produced (output) design object(s) (Fig. 3.9(c)). If the produced design object(s) is(are) required to be updated, the normal way is to perform the automatic transformation process again provided that the consumed design object(s) is(are) updated first. Updating the design object in the editor way through another design session may also be permitted when the current sub-process is followed by an

interactive sub-process, which is used to refine the automatically produced design object. A variant to this case is found with semi-automatic tools which need a large amount of interactions before invoking the last automatic reasoning procedure (Fig 3.9(d)). Intermediate data models to function as SavePoints are required to ensure the resume of a half-done design. As discussed in Section 3.2.1, process re-engineering techniques can be used to move the interactive design tasks upwards to its preceding sub-process, which transforms the case shown in Fig 3.9(d) into the case shown in Fig. 3.9(c).

Therefore, all the engineering data manipulation means can be classified into two types: the *Load-Interactively Operate-Save* mode and the *Load-Automatically Deduce-Save* mode. Identification of the engineering data manipulation means is the foundation to develop an adequate design transaction model through which engineering tools interact with a shared data store monitored by a data manager, such as a PDM module.

3.4.2. Basic Design Transaction Model

Different data managers may adopt different design transaction models. The most widely used design transaction model is called the Check-Out/Check-In transaction model in which, “a design transaction corresponds to the period of time from the Check-Out to the corresponding Check-In” (Wolf 1994), as illustrated in Fig. 3.10. Once checked out by a particular tool via the data manager, the design object in the shared data store is applied a lock. The lock mechanism is implemented in the data manager, which may be a part of a larger integration framework providing more integration functions beyond that of data integration. Re-check-out of this design object is then prevented until it is successfully checked in again and the lock is

removed. The particular tool can then safely operate on a copy of a particular design object in its own working space for an uncertain period of time without worrying about that other users may also check out this design object and generate a conflicted update. It is important to note that a design transaction in this basic design transaction model is performed by a single engineering tool on a single design object and is the basic unit of consistency for operation by engineering tools on design data.

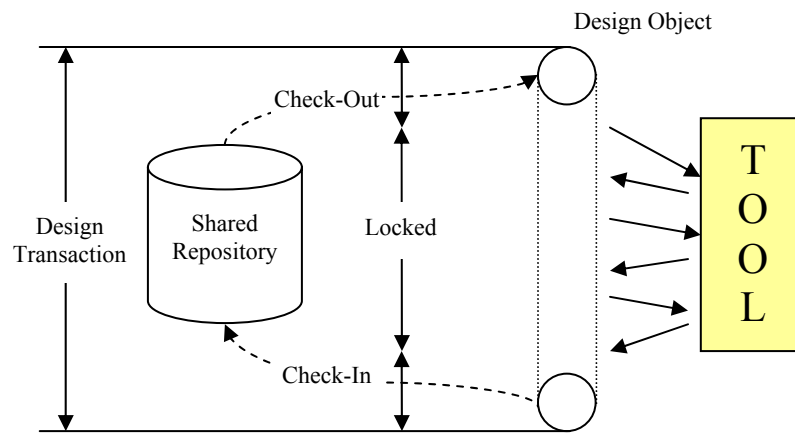


Fig. 3.10. The basic design transaction model (Wolf 1994)

3.4.3. A Special Design Transaction Model for Feature-Driven Engineering Process

It can be easily found that the basic transaction model can only be directly used for the data-tool interaction case shown in Fig. 3.9(a). For the case shown in Fig. 3.9(c), it cannot be used directly. One way to overcome the deficiency of the basic transaction model is to use the workspace concept to collectively treat the consumed data and the produced data as a consistency unit (Katz *et al.*, 1986; Rehm *et al.*, 1988). A design transaction then denotes the manipulation of multiple design objects in a private workspace. Check-Out and Check-In operations are used to transfer design objects to/from the private workspace and the shared archive. Changes made in a private workspace are not visible to other engineers and the original replicas in the shared archive are locked to prevent unmanaged writes. Another similar way is to use the

concept of *complex objects* which contain sub-objects and further represent a design information access pattern (Ranft *et al.*, 1990). The consumed data and the produced data are collectively viewed as a complex object and accessed as a whole when checked out or in. One common problem for these two or other similar proposals is that they all involve a certain extent of unnecessary Check-Out of some design objects. Using the case described in Fig 3.9(c) as an example, suppose the produced design objects need to be updated based on pre-defined updates on consumed design objects. The corresponding design session only requires checking out the consumed design objects. The produced design objects are only required to be locked during the design session without need to be checked out. Check-out of the unnecessary design objects is a conservative strategy and may be tolerable if these design objects are of moderate size. Unfortunately, the feature-driven engineering process may involve produced design objects of very large sizes in some design sessions. For example, in the process to perform the progressive die configuration task with the die operation feature model as the consumed design objects (Fig. 3.7), the produced design objects include a huge set of die configuration descriptions. Therefore, the workspace or the complex objects strategy is not exactly adequate to define a transaction pattern for the case shown in Fig. 3.9(c).

However, a slight augmentation to the basic design transaction model with introduction of three advance concepts can fill the gap and yield a well suitable design transaction model for feature-driven engineering processes. The first two concepts are termed **virtual Check-Out** and **virtual Check-In** as compared to the physical Check-Out and physical Check-In in the basic transaction model. Specifically, the physical Check-Out operation performed at the beginning of a design transaction will produce a

set of copies of the design objects in the private workspace while applying a set of locks on the corresponding design objects stored in the shared repository. However, the virtual Check-Out operation only applies a set of locks on the corresponding design objects without producing physical design object copies in the private workspace. Similarly, the physical Check-In operation performed at the end of a design transaction will overwrite the value of the design objects to be checked in with the new values generated in the private space while removing the corresponding locks on the these design objects. However, the virtual Check-In operation only removes the corresponding locks on the corresponding design objects without overwriting their physical values provided that they are unchanged in the private workspace. In the implementation aspect, the physical-Check-Out/virtual-Check-In transaction equals to a *read-only* operation and the virtual-Check-Out/physical-Check-In means placing a lock on the corresponding design object at one predefined moment till a direct overwrite is performed.

The third concept is termed **transaction group**, a terminology originally used by researchers such as Roller *et al.* (2002a) when dealing with synchronous cooperative work based on a shared engineering database. The grouping criteria here are purposely adjusted to reflect the requirements placed by the feature-driven engineering processes. Specifically, transactions on all the consumed design objects in a design session are viewed as a Transaction Group, which is a logical unit of work, and the transactions on all the produced design objects as well. In this sense, a design session with the working mode as shown in Fig. 3.9(c) involves two design Transaction Groups. The first group consists of a set of physical-Check-Out/virtual-Check-In transactions which are equivalent to *read-only* operations for the consumed design objects and thus can be

grouped automatically. The other group consists of a set of virtual-Check-Out/physical-Check-In transactions for the produced design objects and ensures that those design objects are checked-in correctly and collectively overwrite their counterparts checked-out.

3.4.4. Discussions on the Proposed Design Transaction Model

According to the proposed design transaction model, a feature-driven engineering process may involve two types of design transactions. For highly interactive edit-style tools (Fig.3.9(a)), a standard physical-Check-Out/physical-Check-In transaction model will become effective. For automatic engineering tools (Fig.3.9(c)), a Transaction Group model including virtual Check-Out and virtual Check-In operations will become effective.

It should be noted that the virtual Check-Out/Check-In operations are not performed explicitly by engineering tools like the physical Check-Out/Check-In operations. Instead, they are required to be performed implicitly and automatically along with a group of physical Check-Outs of the consumed design objects with the help of certain managerial tool. Therefore, the primitive operations involved in implementing the augmented design transaction model are still physical Check-Out and Check-In.

The operation logic for the case shown in Fig.3.9(c) can now be clearly described as follows: once the consumed design objects are checked-out and the corresponding automatic engineering tool is initialized, the managerial tool automatically executes the virtual Check-In of the consumed design objects to finalize the design transactions on the consumed design objects within one Transaction Group; immediately after these

operations, the managerial tool executes the virtual Check-Out of the produced design objects and initializes a set of transactions on these design objects within another Transaction Group; this transaction Group is then terminated by the corresponding Check-In of the newly produced design objects by the automatic engineering tool.

This operation logic clearly shows that the augmented design transaction model outperforms the workspace model in that it removes unnecessary operations to physically check-in the consumed design objects and check-out the produced design objects in a tool-run. The former can be removed because they are unchanged in the tool-run. The latter can be removed because they are not physically consumed by the tool and then unnecessarily present in the private workspace.

CHAPTER 4

OVERVIEW OF THE CAX FRAMEWORK-BASED INTEGRATION APPROACH

As stated, while there exist considerable conceptually potential integration approaches to be followed to build up a network-integrated engineering environment, this study favors the CAX framework-based approach due to the unique integration power of the CAX framework concept. With the help of this concept, the challenge to develop facilities to conveniently integrate multiple CAX tools into a coherent engineering environment can be overcome by introducing a common CAX framework for these tools. The components that are desirably incorporated into the framework can be easily identified and specified. It also leaves a large space for system developers to selectively and adaptively use those formulated components and services so that the framework can behave in a particular way compatible to a set of pre-specified requirements. This allows a special framework to be devised to provide the end-users with supports in sharing common information, process management, *etc.* This chapter presents an overview of the CAX framework-based integration approach, setting up a basis to develop advanced integration functions for feature-driven engineering processes. Those functions include the unique version/configuration and process

management services, which explicitly take into account the identified characteristics presented in the previous chapter.

4.1. Rationale of the CAX Framework Approach

The most prominent characteristic of the CAX framework approach is that it transforms the complex tool integration missions into a definite process to develop a CAX framework for the distributed CAX tools or tool users. This section explains why the CAX framework concept can be employed to develop the desired integration facilities for the tools involved in a specific application domain, such as the feature-driven engineering process. Basically, the integration power of the CAX framework is attributed to the roles that can be allotted to it.

Analogous to the CAD framework (Wolf 1994), three basic roles can be allotted to the CAX framework so that an integrated engineering environment can be achieved. These roles are that of common product data repository, engineering data manager and engineering process manager. By playing the first role of a common product data repository, the CAX frameworks ensure that all the engineering data generated by the CAX tools is centrally stored in the common product data repository. It is thus possible to avoid the data redundancy and inconsistency problems which are always encountered in the product development process using a set of completely isolated engineering tools.

The second role that can be allotted to the CAX framework is that of engineering data manager similar to a PDM module. By playing this role, the CAX framework can capture the global view of all the engineering data that are generated by dispersed

CAX tools and support versioning control and configuration management to improve data integrity and consistency. By nature, an engineering data management system is always attached to a common product data repository and a common product data repository always coexists with a management system. It is therefore understandable that the above two roles may be collectively referred as one role called a product data manager.

The third role that can be allotted to the CAX framework is that of engineering process management similar to a WM module. By playing this role, the CAX framework can provide a design flow browser which enables the designer to inspect the status of his design and to invoke the right tools. Incorrect tool execution sequence and misuse of data sources can be avoided. Correct tool and data source selection can be rapidly identified without the need of extra efforts to search information that is unorganized.

Due to its flexibility, extensibility, modularity, portability, and maintainability, the CAX framework may be allotted with additional roles apart from the above three basic roles inherited from the CAD framework concept. Some functions described in other relevant literature pertaining to a network-integrated engineering environment (see Chapter 2) may be implemented in the CAX framework. For example, the common product database can be extended to become a knowledge base or a knowledge repository through enlarging its database structures to richer representation schemata (Roller & Eck 1999). Inference facilities are accordingly added to provide more intelligence and active behavior to the database system at the same time. By using this intelligence and active behavior, two types of assistance can be attained to make an engineering process more productive and less error-prone. One of them is related to

reuse of the former good engineering designs stored in the product database to develop new analogous engineering designs by using an approach called Case-Based Reasoning (CBR) (Tor *et al.*, 2003). The other is related to situation detection, semantic integrity enforcement, concurrency control, collaboration support, and storage management by using active database management systems (Roller & Eck 1999). Therefore, the CAX framework can be allotted with a role of common knowledge repository for the end-users and the participating tools. Any other types of knowledge shared by the CAX tools (especially some intelligent CAX tools with a knowledge base attached) can also be centrally managed in this knowledge repository.

Another possible design choice to make a CAX framework more powerful is to incorporate the CSCW service like that in the CONCERN architecture (Hanneghan *et al.*, 1995, 1998) into it. This makes it possible to allot the CAX framework with a role of a CSCW service provider. Yet another possible design choice is to incorporate any sharable service, such as the geometric modeling service (Shah *et al.*, 1997) into the CAX framework, which makes it possible to allot the CAX framework with the role of a common geometric modeling service provider. Participating CAX tools can then invoke this service once the need arises. In summary, any common services which are shared by multiple distributed users or client-side CAX tool applications can be incorporated into the CAX framework. Certain corresponding roles can then be allotted to the CAX framework.

Once the CAX framework that can be allotted with the above roles is incorporated into the engineering environment, the CAX tool users can exploit both the dedicated functionalities provided by the tool they are using and the common integration

functions provided by the CAX framework. The tools can still run autonomously, but the design activities carried out on these tools are (semi-)automatically coordinated with the help of the CAX framework.

4.2. Definition of Functional Requirements and System Architecture

With the confirmed confidence of the integration power of the CAX framework concept, the emphasis is now put on how to develop a CAX framework for the feature-driven engineering processes, such as the progressive die design and manufacturing process. There are a number of strategies that are used to define the functional requirements of the framework and the general system architecture. Some of them are adapted from those that are developed by the CAD framework researchers in the field of EDA. Others are developed from the beginning to address the domain-dependent issues involved in the development of the CAX framework, which aims at applications in the field of manufacturing engineering.

4.2.1. Functional Requirements

Definition of the functional requirements of the CAX framework needs to consider what integration functions are desirable by the CAX tools as well as the tool users and what functions the CAX framework can provide. Section 4.1 has shown that the CAX framework can be allotted with diverse roles as mentioned above. However, it is found that what are most important for the system integration from design to manufacturing for the feature-driven engineering processes are still the three basic roles: common product data repository, engineering data manager and process manager, or simply the latter two roles. Therefore, the functional requirements of the CAX framework are

defined as providing product data management services and engineering process management services for the CAX tools and tool users.

In this sense, the CAX framework can be compared to a lightweight PDM module combined with a WM module, which co-works with the CAX tools involved. As a lightweight module, it requires less demanding computing resources and no excessive system customization operations are required before running the system. It is possible that the CAX framework functions can be provided by customizing a heavyweight PDM/WM system. However, the customization approach is inferior because of the reasons presented in Chapter 2.

4.2.2. Some Basic Strategies for Defining the General Framework Architecture

The integration functions of the CAX framework are exactly the same as that of the CAD framework (Wolf 1994). However, due to the different working modes of the tools that interact with the framework and the different structure of the tool data that will be centrally managed by the framework, the internal structure of the CAD framework and the CAX framework would be different. Despite the differences, the effectiveness of some basic strategies for defining the general framework architecture (Wolf 1994) still holds. Some of these strategies are presented next.

- Split the Framework into Framework Kernel and Workbench (Framework Tools)

Apart from providing interfaces for the participating CAX tools, the CAX framework should also provide interfaces to the end-users so that they can be informed about the status of his design or initiate some framework actions. The framework is then split into two parts, the framework kernel and framework tools, the latter of which aims

specifically at interactions of the end-users with the framework. In manufacturing engineering, one may notice that the “workbench” module in many architecture (Hanneghan *et al.*, 1998; Conaway 1995) takes the same responsibility as the framework tools. A workbench or workbench application is a common user interface to multiple applications used within a particular discipline. It provides a graphical front-end to the users so that they can access the services of the environment. Therefore, the CAX framework in this thesis is characterized by consisting of a workbench and the CAX framework kernel.

- Separate Meta Data and “Raw” Engineering Data Handling

There are two types of data that are maintained within the framework, the actual design data and the meta data which means “data about data”. The meta data owns pointers pointing to the design objects and are used to index the design data as well as to apply management strategies. Separation of meta data and “raw” engineering data implies that the framework kernel should contain two built-in databases, the meta data database or management database and “raw” engineering data database or design object repository, to accommodate them respectively. The meta data is small in storing size compared to the volume of the corresponding “raw” engineering data. Further, the collection of the meta data in a project is of complex structure and a dedicated database management system with specially designed schema is required. On the other hand, the collection of the “raw” engineering data in a project is simply a collection of design objects which are identified by its file name. A part of a file system is then sufficient to take the responsibility to function as the design object database for a project.

- Treat the Framework Kernel as Transaction Processing System

In the CAX framework kernel, any management service requests from the CAX tools (tool wrappers) or the workbench are eventually responded with certain operations to consult or update the administered state of the meta data or the engineering data. As stated, the meta data is stored in the management database and the engineering data is stored in the design object repository. It is therefore possible to use the transaction concept, which is widely used in the design of database management systems, to characterize the CAX framework kernel as a transaction processing system. A transaction is a sequence of operations that is either performed completely or not at all. It is a logical unit of work, which transforms a consistent state of the database into another consistent state (Gray & Reuter 1993). The use of the transaction concept provides convenient means to solve the problems of concurrency and recovery. Corresponding to the separation between the meta data and the engineering data handling, two main types of transactions are involved in the CAX framework kernel: meta data transactions and engineering data transactions. The meta data transaction carries the identical semantics of the conventional transactions. The design transaction is semantically different from the conventional transactions and relates to the way how the engineering tools manipulate engineering data. For the CAX framework dedicated to the feature-driven engineering process, the special design transaction model presented in Section 3.4.3 should be applied. The meta data transaction can be further classified into project transactions, configuration transactions, *etc.* These transactions should be properly layered (see Section 5.2.1) and coordinated to make the design data correctly checked out from and checked in to the design object repository. Treating the framework kernel as a transaction processing system justifies the strategy for the framework to adopt a standard database to store the meta data and a directory of file system to store the design objects.

4.2.3. The General System Architecture

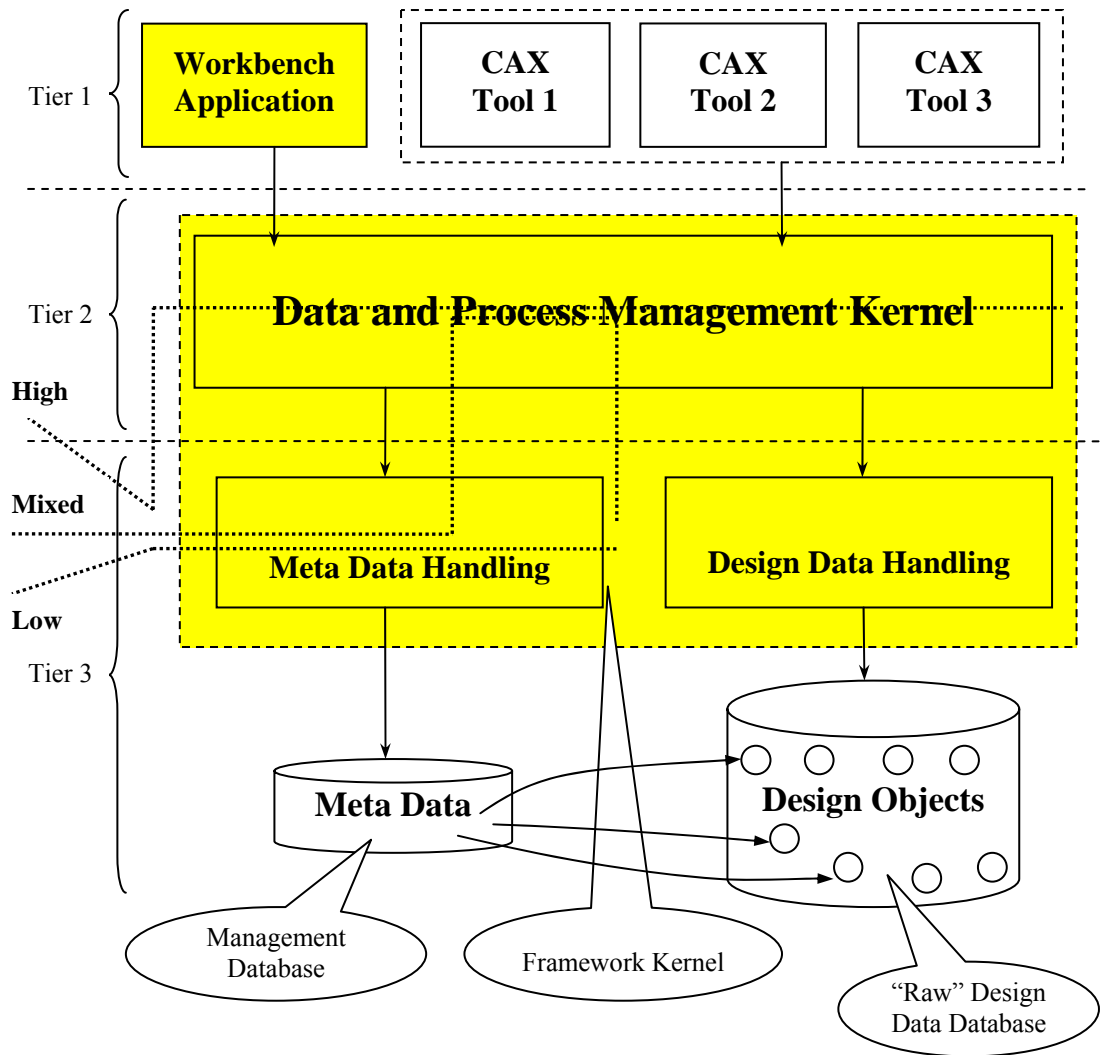


Fig. 4.1. The general system architecture (Wolf 1994)

By using the framework design strategies presented above and consulting the CAD framework architecture described by Wolf (1994), the general system architecture of the CAX framework is defined in Fig. 4.1. In this architecture, the overall CAX framework-based engineering environment consists of the CAX tools and the CAX framework. The framework further comprises the workbench application, the framework kernel and two data stores, the management database and the design object repository. The management database stores meta data, which owns pointers pointing

to the design objects stored in the design object repository. The workbench application interacts with the framework kernel directly and the CAX tools via wrappers (see Section 4.4.2). The framework kernel is further decomposed into three components, the data and process management (DPM) kernel, the metadata handling component and the design data handling component. The overall architecture complies with the popular 3-tier strategy for the development of the Internet-based applications. The CAX tools and the workbench are the first tier or the user interface tier. The framework kernel is the middle tier or the logic tier. The management database and the design object repository are the third tier or the application data tier.

4.3. A Roadmap of Implementation and the “Skeletal” Framework

This section overviews the main steps that are taken to develop the CAX framework up to the physical level beginning from the functional requirements and the general system architecture defined in the previous section. The “skeletal” framework, which refines the general system architecture by considering functionality partition between the server side and the client side, is also presented.

4.3.1. A Roadmap of Implementation

Wolf (1994) recommended developing a CAD framework via three main steps. The first step is to develop the “information architecture” that defines the information structure of the framework. The second step is to develop the “component architecture” that identifies the individual framework components and the dependencies between them. The final step is to develop the implementation architecture to define the internals of the framework at the physical level. It is found that this three-step approach is only appropriate without applying the OO concepts. If

the OO concept is incorporated, development of the information architecture and that of the implementation architecture can be combined in one run, because they are both the task to identify a set of object classes and their relationships. Further, it is found that most of the details of the component architecture can be specified without the need to know every detail of the information architecture. Therefore, it is decided to develop the current CAX framework through two steps. The first one is to develop a “skeletal” CAX framework up to the physical level. All the framework components are specified either using existing software products or are developed from the beginning. The second one is to develop an adequate schema for the management database and the information architecture for the components that need to be developed from the beginning. All the required user operations are also defined in this step so that the GUI can be easily devised.

Of the above two steps to develop the CAX framework, the first one is relatively easy and the results are presented in the rest of this chapter. The second one is the most creative and challenging part of this thesis. It involves modeling and analyzing the desired engineering environment in two aspects , the product data management aspect and the engineering process management aspect, the latter being extended from the former. The next two chapters are respectively dedicated to deal with these two aspects to show how the unique product data management and engineering process management functions are incorporated into the CAX framework. Note that the system development route adopted in this study while using the CAX framework integration approach may be duplicated to develop a similar network-integrated engineering environment for applications in other domains.

4.3.2. Functionality Partition between the Client and the Server

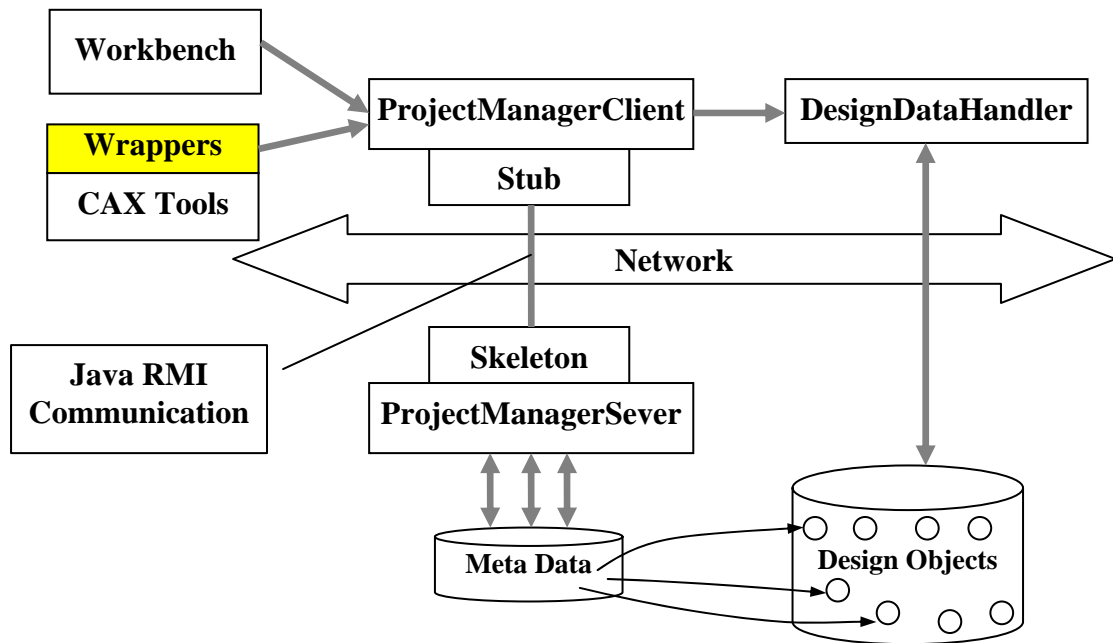


Fig. 4.2. Creation of Client/Server with Java RMI

To make the general system architecture shown in Fig. 4.1 more specific so that a “skeletal” framework is defined, it is desirable to first partition the functionality between the client and the server. There are basically three possible levels at which to define the client/server boundary between a client process and a server process. Firstly, the client/server boundary may be defined at the highest level as indicated by the dotted line tagged with ‘High’ in Fig. 4.1. Only the user interface applications including the workbench and the CAX tools reside on the client side. This is a design of thin client/fat server. The other extreme is to place the client/server boundary at the lowest possible level as indicated by the dotted line tagged with ‘Low’ in Fig. 4.1. Only the management database and the design object repository as well as the corresponding data handlers reside on the server side. This is a design of fat client/thin server. These two extremes overload either the server or the client. The optimal solution, therefore, is to place the client/server boundary at the medium level as indicated by the dotted line tagged with ‘Mixed’ in Fig. 4.1. After re-grouping the

components on the client side and the server side based on this mixed approach, a “skeletal” framework is gained as shown in Fig. 4.2. Some details are depicted next.

According to the mixed approach, the server object is mainly responsible for Meta Data Handling and a part of DPM functions. This combined server object is now defined as ProjectManagerServer. The rest part of the DPM functions is collectively defined as ProjectManagerClient object which is located at the client side. Apart from the ProjectManagerClient object, also located at the client side are the workbench application and the engineering tools, which directly interface with the users, and the Design Data Handler, which is responsible for design data access and implemented based on the jCIFS open source client library (see Section 4.4.4). The DPM functions cannot be entirely allotted to the ProjectManagerServer because the Design Data Handler, which is frequently requested by the DPM, is located at the client side, otherwise, the communication overhead will be increased. User interface applications cannot directly request methods in the ProjectManagerServer object but through the ProjectManagerClient object which carefully sequences the operations on the meta data and design data. The ProjectManagerClient object calls the methods within the remote ProjectManagerServer object through a ProjectManager interface which is implemented by the ProjectManagerServer. The Java RMI communication facilities including the stub object on the client side and the skeleton object on the server side physically realize the client/server communication. The calling dependency and the creation of the client/server with Java RMI are elaborately illustrated in Fig. 4.2.

4.4. Some Basic Implementation Decisions for the CAX Framework-based Network-integrated Engineering Environment

To implement a network-integrated engineering environment physically, a range of design choices should be made to define the system environment and to identify appropriate computer tools used in the system development process. This section presents some basic system implementation decisions with respect to development of an enterprise-affordable system using IT technologies and software products available at the time. It is obvious that these implementation decisions are not given as an only solution to the related implementation issues. Rather, multiple solutions are possible. Efforts have been devoted to optimize the current solution as satisfactory as possible among the alternatives available. However, the solution should evolve over time.

4.4.1. Platform and Programming Language

The platform is the basis of the entire environment, including the hardware and the operating system software, on which the framework and the tools are to run. Considering a range of factors, such as transparency of distribution and multi-user support, an enterprise-wide Microsoft® Windows-based Intranet is supposed to be the normal working platform. Design engineers from different departments can participate in the common network-integrated engineering environment to carry out a project smoothly. Working at home or in travel to access the centrally-managed data through the Internet is permitted.

Despite its complexity, the CAX framework can be viewed as a programming model on top of the system environment and common basic services to unify the engineering tools with the meta data and design object repository. A common programming language, Java, is selected to define this model in this study. Selection of Java is appropriate since Java is an OO language with client/server capabilities running on the

JVM (Java Virtual Machines) available on the Microsoft® Windows platform. Further, remote communication between programs written using the Java programming language can be easily realized through the Java Remote Method Invocation (RMI) mechanism. The Java RMI system allows an object running in one JVM to invoke methods on an object running in another JVM. This is the reason why the Java RMI is used for creation of the client/server for the CAX framework in the current prototype implementation.

4.4.2. The Wrapper and the Way to Make the CAX Tools Available on the Internet

From the perspective of the end-users of the CAX tools, the CAX framework introduces three new operation types for them, *i.e.*, browsing the design states (dynamic workflows and versions/configurations) maintained in the meta data database, check-out design objects from and check-in design objects to the shared data store across the network. It is easy to understand how to make these functions available in the workbench application because it is an internal part of the framework and developed from the beginning along with the other components within the CAX framework. Particularly, the workbench application is a GUI that communicates with the CAX framework kernel through an interface from which to retrieve meta data and check-out/check-in design objects (arrow 1 in Fig. 4.3). However, for the CAX tools, given that they are legacy applications that work independently, extra efforts are required so as to make the above functions available to these tools or to make the tools available to the CAX framework and thus on the Internet. Basically, there are two ways, the indirect way (arrow 4 in Fig. 4.3) and the direct way (arrow 3 in Fig. 4.3), for the CAX tool users to call the CAX framework functions.

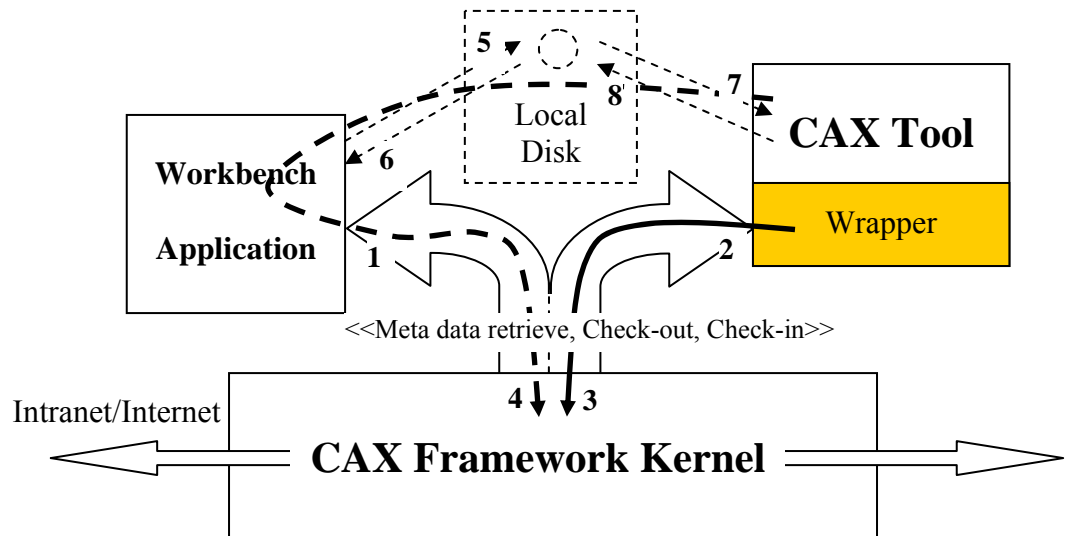


Fig.4.3. Integrating a tool with the framework kernel through a wrapper

In the indirect way, the user has to interchangeably work in the CAX tool environment and the workbench application environment. If he wants to browse the design states, he must leave the CAX tool environment and launch the workbench application that consists of a workflow browser and a version/configuration browser. If he decides to work on a certain design task and wants to check-out the corresponding design object(s), he first locates that design object(s), checks it (them) out and temporarily saves it (them) in the local disk with the help of the workbench application (arrow 5 in Fig. 4.3). Then, he returns to the CAX tool environment and opens the temporarily-saved design object(s) (arrow 7 in Fig. 4.3). If he has finished a task using a CAX tool and want to check-in a (set of) design object(s) to the shared data store, he first saves it (them) in the local disk (arrow 8 in Fig. 4.3) and then accomplishes the check-in operation with the help of the workbench application (arrow 6 in Fig. 4.3). Apparently, the indirect way is not “transparent”. The required operations are error-prone and inconvenient for the end-users.

In the direct way, the user can achieve the above work mode without the need of either leaving the CAX tool environment or using the local disk to temporarily save the relevant design object(s). This is enabled by the wrapper that wraps the engineering tools and integrates them into the CAX framework to form a more tightly-integrated engineering environment. From the perspective of the CAX tool users, introduction of the wrapper means adding some extra menu items or buttons within the existing tool GUIs. Activating these menu items or buttons will further activate windows for the users to browse design states and check-out/check-in design objects. From the perspective of the system architecture, the wrapper is a specially written software layer which intercepts and re-routes commands issued in the tool application environment to call the services provided by the CAX framework kernel. The CAX tools can thus communicate with the CAX framework kernel so as to be available on the Internet bypassing the wrapper that is connected to the same interface as provided to the workbench application (arrow 2 in Fig. 4.3).

The main challenge to implement a wrapper is to solve the relevant interoperability problems between the different programming languages used by the wrapper and the external application. In the current implementation test, the CAX tools were built on top of AutoCAD® which provides application developers with an Application Programming Interface (API) called ObjectARX®. ObjectARX® is a very powerful C++ runtime extension programming environment which allows external applications to execute operations on AutoCAD's or the CAX tool's data and monitor user functions, such as 'saving' and 'loading' of designs. Written in C++ language, the wrapper can call the methods in the CAX framework kernel, which is written in Java. For example, this website, <http://www.javaworld.com/javaworld/javatips/jw-javatip17->

[p2.html](#), has explained in detail the way to call Java methods from C++. Other types of CAX tools if made to participate in a certain similar CAX framework can also use the above wrapping technology. For example, the Pro/ENGINEER® CAD system provides an API called Pro/DEVELOP®, which functions like the AutoCAD's ObjectARX®.

The tool wrapping mechanism allows the prototype implementation to be conducted incrementally. The first step can temporarily overlook the procedure to physically wrap the CAX tools to integrate them with the framework kernel, and uses the indirect way to test the framework functions bypassing the workbench application and the local disk. The second step is to incorporate the tools into the integrated environment through the wrapper based on successful development of a CAX framework. By isolating the development of the framework from that of the wrapper, the complexity of the required effort is significantly decreased.

4.4.3. DBMS for the Management Database

Sharing the meta data which is common to a number of CAX tools is the basis of a CAX framework-based engineering environment. It is expedient to use an OO database management system (OODBMS) as the meta data storage system for several reasons. Firstly, OODBMSs are well-suited for engineering applications due to their rich modeling power through the concepts of classification, inheritance, generalization and aggregation. Further, not only the state of the real world entities can be described using the attributes, but also the behaviors using the methods in the class definition (Ramakrishnan & Janaki 1996). Secondly, OODBMSs offer significant flexibility for handling highly interrelated data of different granularities on which different types of

access are performed. Thirdly, using OODBMS is in keeping with the OO nature of the entire CAX framework design thereby ensuring consistency throughout the project. The object management layer programmed in a certain programming language, such as Java, can create, load, delete, and store objects and further invoke their methods (Hanneghan *et al.*, 1995, 1998).

A number of OODBMSs have been used as the basic component of some integrated engineering environments. For example, OBST, an OODBMS freely available, has been used for the repository support service in the CONCERT environment (Hanneghan *et al.*, 1995, 1998). In another example, ObjectStore® was chosen to integrate concurrent design processes with respect to storing design data, sharing design information, recording experience and increasing data reusability. Version management and schema evolution on top of this OO database system were also discussed (Hsiang *et al.*, 1999). In another example, an OO database system, also ObjectStore®, was used to store VRML (Virtual Reality Modeling Language) objects so that they can be shared and updated by multiple users in real-time. Concurrency control mechanisms of the system were utilized to deal with the concurrency issues arising from simultaneous updates (Turgut *et al.*, 2001).

Based on a thorough review of the standard OODBMS products which were commercially or freely available, the current prototype implementation selected ObjectStore® to store and manage the meta data in the CAX framework.

4.4.4. File Transfer

In the CAX framework, the meta data transactions manage the operations on the design data which are treated as large files with its own native data structure. Upon execution of an operation, such as creation, update or removal of a design object, the transfer of design object files takes place along with transfer of corresponding meta data within a series of carefully sequenced procedures. Section 4.2 has shown an approach to transfer meta data as parameters in RMI. This approach cannot be used for the transfer of design objects because they are large files and the transaction time associated with this may not be acceptable. One popular acceptable way to implement file transfer between two distributed locations is to automatically invoke an ftp operation external to RMI, simply using RMI as a notification mechanism. The server's computing work can then be reduced because the framework server can focus on meta data operations. The disadvantage of this approach is the lack of code portability and the need for an additional ftp server which further needs a common directory to store the files "ftped" from the client, as well as the error log files created by the file transfer procedure (Urban *et al.*, 1999b). The current study uses a more flexible and Java-compatible file transfer approach based on the CIFS (Common Internet File System) (Leach & Perry 1996). jCIFS SMB* client library (Anonymous 1), which enables any Java application to remotely access shared files and directories on SMB file servers (*i.e.*, a Microsoft® Windows "share"), is used to develop Java-based client applications. A small amount of customization operations on the server side is needed given that the CIFS server is a built-in component in most Microsoft® workstations. Since SMB file servers on UNIX systems are also available, this approach is scalable to multiple computing operation platforms.

* SMB or **S**erver **M**essage **B**lock protocol is the file-sharing protocol at the heart of CIFS and thus the CIFS servers (clients) are also called SMB servers (clients).

CHAPTER 5

VERSION CONTROL AND CONFIGURATION MANAGEMENT

Chapter 3 has identified a comprehensive set of characteristics for a feature-driven engineering process. Chapter 4 designed a CAX framework-based network-integrated engineering environment with the management database schema and the corresponding user interface design being left open. From this chapter onwards, issues related to filling these openings will be addressed. The final integration functions provided by the developed engineering environment will be fully described. The most important and challenging procedure involved is information modeling which would comprehensively take into account the outcomes gained in Chapter 3. As explained above, the information modeling process is performed incrementally. This chapter focuses on the product data management aspect and a unique version control and configuration management model for feature-driven engineering process is presented. The next chapter focuses on process management aspects.

5.1. Version Control and Configuration Management Concepts

One of the main aims to integrate a range of engineering tools to form a coherent environment is to offer a uniform repository in which all data are stored and shared (Bounab & Godart 1998). Therefore, integrated engineering environments invariably

involve dealing with the management of a large number of different kinds of design objects created throughout the development life cycle. According to Chapter 2, providing such a function is attributed to the PDM or data integration mechanism. The main issues which are related are version control and configuration management. A brief introduction for this concept is first presented next.

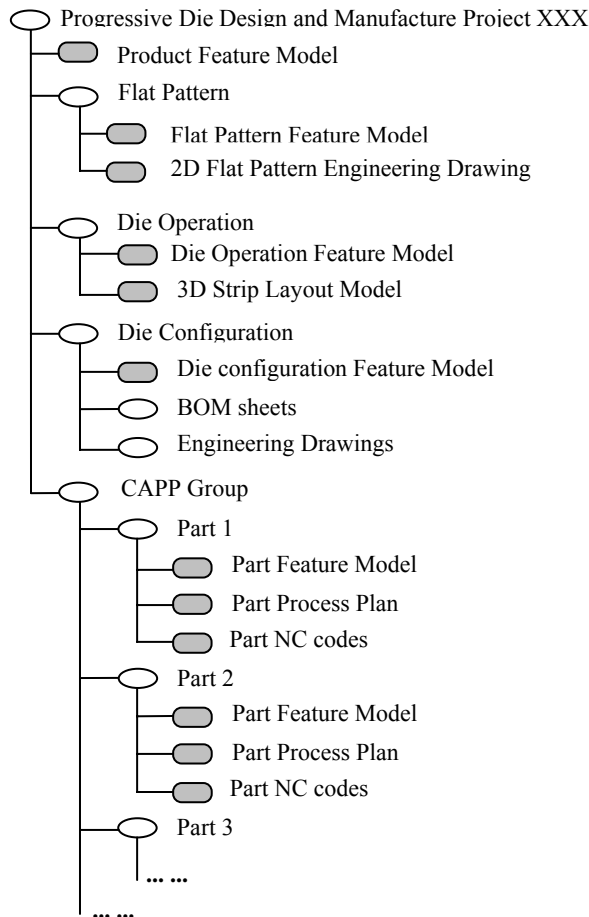


Fig. 5.1. Product configurations

To make navigation easy, the tremendous amount of design data in the shared repository must be organized along certain dimensions dependent on different application contexts (Katz & Chang 1987). In general, all the data that describes the same physical entity should be organized such that it can be treated as a collection and the collections are in turn arranged into a hierarchy of directories. The state of a complete design object hierarchy is referred to as a configuration. Fig. 5.1 shows an

example of a product configuration for the design data structure involved in an integrated progressive die design and manufacturing process. If all the data and the configuration are of mono-version, the repository strictly maintains the latest state of every design object and the configuration with the last state introduced by the user will always replace the previous one. A network file system is probably sufficient to implement the repository in this case and no special product data management assistance is needed as long as the repository is accessible by distributed users and the problem of write contradictions to the same documents is resolved.

However, the iterative and exploratory nature of the engineering process prompts the designers to generate and experiment with multiple alternative descriptions of a design before selecting one that satisfies the design requirements (Ahmed & Navathe 1991). The configuration is also treated as a versioned object and more than one configuration can coexist (Agrawal & Jagadish 1989). Selecting a version for each design object that constitutes the configuration is referred to as configuration management which should guarantee that the desired relationships such as “*is-derived-from*”, “*is-a-component-of*” and “*is-dependent-on*” are correctly maintained. Most of the past versioning solutions mainly concentrated on using certain references to correlate design object versions, as well as configuration versions (Beech & Mahbod 1988; Ramakrishnan & Janaki 1996; Miles *et al.*, 2000; Carnduff & Goonetillake 2004). A reference to another version from within one version is also called a binding, which can be further classified into static and dynamic binding (Carnduff & Goonetillake 2004). Binding mechanisms are successfully used to deal with derivation (between different versions of the same design object) and composition relationships. However, few binding mechanism-enabled versioning schemes capture design semantics such as a dependence of a manufacturing representation on its upstream product definition. Some proposals

(Baldwin & Chung 1995; Ramakrishnan & Janaki 1996; Westfechtel 2000) did provide certain assistance in the management of design semantics-oriented dependence relationships, but they provide no means to control the design change propagations while comprehensively taking into account the design change propagation properties inherent in the engineering processes. With the intention to overcome this deficiency, the next section presents a special version and configuration management model making the most of the identified characteristics of the feature-driven engineering process (Chapter 3), especially in the aspect of design change propagation properties.

5.2. A Version Control and Configuration Management Model for Feature-driven Engineering Processes

This section addresses the version control and configuration management issues relevant to development of the product data management functions in the integrated engineering environment for the feature-driven engineering process. There are some existing solutions for some of these issues, and accordingly, they are adaptively adopted in this study. For the other issues that are mainly resulted from incorporation of the design change propagation properties presented in Chapter 3, special solutions are developed.

5.2.1. Basic Concepts

The versioning model developed in this study makes use of some essential concepts found in the database version approach (Ahmed & Navathe 1991) with respect to the set-up of the basic version control and configuration management framework. Some adaptations are made accordingly based on the identified design change propagation properties. Design objects with organizational information including “structural

objects” (Katz & Chang 1987) are managed by a multi-version database, which is defined as a set of logically independent and identified database versions (DBVs) (Fig. 5.2). Each DBV contains an exact configuration consisting of one version of each constituent object. Both the database version and the configuration version thus refer to the same thing and both terms are used interchangeably here-in-below. Version control both at the design object level and the configuration level is supported by a set of operations on the DBVs. Formally and fundamentally, a DBV is defined as a tuple composed of the DBV identifier and the set of versions of all the objects contained in the multi-version database, one version per object (Ahmed & Navathe 1991). This definition is further semantically augmented in this study to incorporate an additional information element, called version annotations which are explained later.

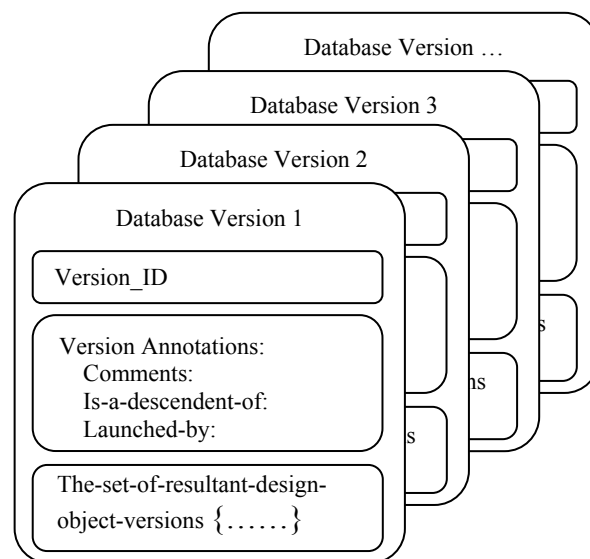


Fig. 5.2. Multi-version database as a set of database

The concept of database versions allows the use of another type of transactions different from the design transactions discussed in Chapters 3 and Chapter 4. This type of transactions, called configuration transactions, logically partitions arbitrary operations on a set of database versions (not design objects) into atomic units of work to transform each database version from a consistent state to another consistent state.

Incorporation of the configuration transaction concept makes the layered transaction scheme discussed in Chapter 4 (page 100) consist of five layers respectively, tool execution, project transaction, configuration transaction, design transaction and design data operation. Each layer, apart from the uppermost one, is wrapped in the layer that is immediately above it by two operations to initiate and terminate it. For example, the project transaction wraps the configuration transactions, which further wrap the design transactions. It is important to note that the Check-In action of a design transaction may recursively refined by another versioning configuration transaction again (the dashed box in Fig. 5.3). Some properties of the configuration transactions are presented next.

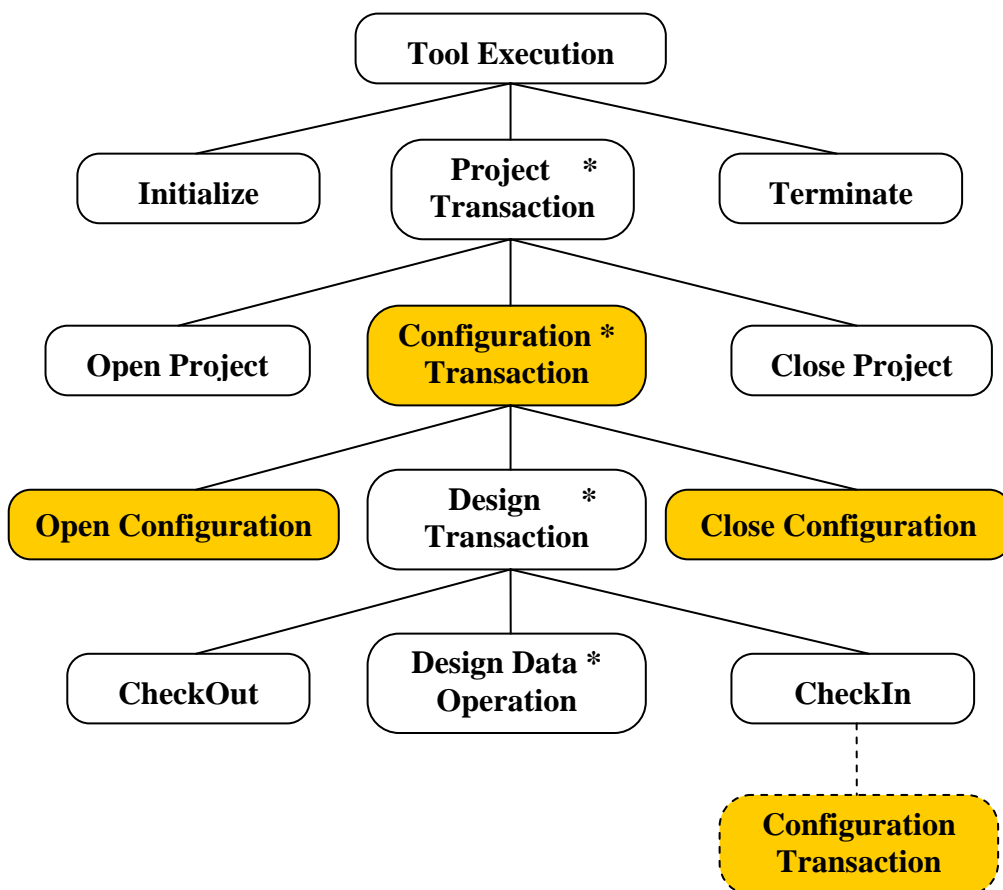


Fig. 5.3 Augmented layered transaction schema for handling engineering data (developed based on Fig. 6.6 in Wolf(1994))

In the simplest case, a configuration transaction concerns one database version and may be non-versioning or versioning. A non-versioning transaction queries or updates a database version, causing it to evolve independently of the other database versions. On the other hand, a versioning transaction creates a new database version from a parent database version. A user operates on the multi-version database in the following way. Firstly, he chooses a database version via the database version identifier, which may be system-generated or manually specified by the user through a versioning transaction at its creation time. When the database version is chosen, the user may perform non-versioning transactions as if he works on a non-versioning database. The system will automatically identify object versions belonging to the database version chosen to provide desirable information to the users in relevant browsers or accept proper updates issued by the users. The user or the user application may also perform a versioning transaction to create a new (child) database version and then work on it. He may further work simultaneously on several database versions, embedding operations that are addressed to different database versions into a grouped transaction. The only requirement is that this grouped transaction must transform all the database versions accessed from one consistent state into another. To sum up, there are two levels of operations on a multi-version database to control versions and configurations. At the upper level, the user or the user application creates and deletes a specified database version or configuration versions. At the lower level, he reads, writes, creates and deletes a specified object in a specified database version representing a configuration.

5.2.2. Design Change Propagation Scope and Object Version Identification

In the course of the development of representations (design objects) involved in a feature-driven engineering process, once a new design object version (causal version)

is created, the changes made to the value of this object may require the value of some other objects to be changed consequently due to the dependency relationships. New versions (resultant versions) with the required resultant design changes being incorporated have to be created. Additionally, there may be some design objects that are not affected by the causal design changes and their new versions (unaffected versions), if created to guarantee data integrity, carry values identical to the old ones according to the design change property described in Chapter 3. The causal, resultant and unaffected design object versions belonging to a configuration should be correctly aggregated to form a new configuration version (or DBV). There are basically two versioning approaches to accomplish this aggregation process.

To understand these two versioning approaches, consider an imaginary feature-driven feature process generating configurations that can be viewed as the variants of a configuration template shown in Fig. 5.4(a). In this figure, square objects *d* and *e* are structural objects, which have only object identifiers. Circular objects *a*, *b* and *c* are physical design objects, which have an object identifier and a value. *e* is composed of *a* and *d*, which is further composed of *b* and *c*. *c* is dependent on *b*, which is further dependent on *a*.

In the most popular versioning approach, the bottom-up approach (Westfechtel 2000), is used, the configuration is treated as a composite object, a new version of which is created in a bottom-up way beginning from the creation of the leaf component object versions (Fig. 5.4(b)). Supposing there is a given set of object versions *a1*, *b1* and *c1* that are mutually consistent to belong to a configuration *Con*, a change made to *a1* leads to generation of object versions *a2* directly and *b2* indirectly. *c* is not affected by

this change and no object version $c2$ is generated. All object versions $a1$, $a2$, $b1$, $b2$ and $c1$ are put into an uncontrolled version pool, from which $Con1$ and $Con2$ are constructed by manually selecting corresponding object versions through a binding process to define the desired composition and dependence relationships. Specifically, the new configuration version $Con2$ is defined to be composed of $a2$, $b2$ and $c1$. Further, $c1$ is dependent on $b2$, which is further dependent on $a2$ in this $Con2$. All these composition and dependence relationships are specified using explicit references to associate the interrelated object versions (Westfechtel 2000). The main drawback of this bottom-up approach is that the operations required can be cumbersome and error-prone, especially for a complex feature-driven engineering process with many unaffected versions, which may recursively inherit value from upper-level unaffected versions.

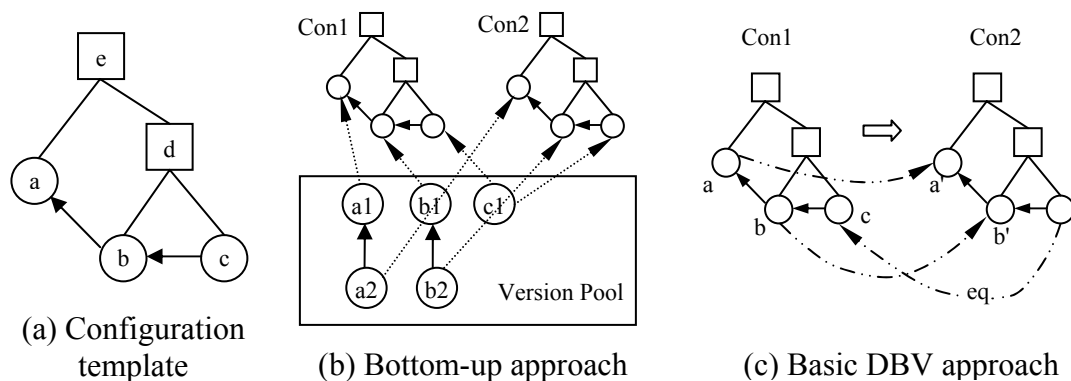


Fig. 5.4. Comparison of two versioning approaches

In order to overcome the deficiency of this bottom-up approach, the DBV approach controls the configurations in a top-down means (Fig. 5.4(c)). Before the entire object versions constituting a configuration are physically created with a valid value, the configuration version is created, or more precisely “pre-created”, in advance by a versioning transaction with a *nil* value being assigned to each constituent object version. The object versions are then read and updated with a valid value through non-

versioning transactions. Compared with the bottom-up approach, no explicit component object versions are created outside the control of the configuration versions using the DBV approach. The template (or a set of configuration rules in the broad sense) allows the structure of a new configuration version to be created before all its constituent objects are created with a physical value. It is the subsequent “value assignment” process, not the binding process, to make *e.g.* *Con2* completely constructed in a way to perform all the desired value changes ($a \rightarrow a'$, $b \rightarrow b'$ and $c \rightarrow$ unchanged) compared to *Con1*. However, due to the existence of the unaffected versions in a configuration version, versions of the same object contained in different database versions may have identical values. If the relevant object versions are copied each time to create a new configuration version, large data redundancy will occur. An ideal way is to let the relevant object value be physically shared by several configuration versions. In this case, not all object versions can be uniquely denoted by a pair: its object name and a new version number distinct from its old one. A special mechanism is required to correctly associate the database version identifiers with its constituent object version identifiers when a new database version is created. Ahmed & Navathe (1991) used a set of dedicated database version stamps to construct the database version in such a way that it is possible to identify all the database version’s ancestors. However, this approach only dealt with the identification problem in the presence of composition relationships without taking into account the dependence relationships. Further, the identifier resolution process is still comparatively complex.

An augmented DBV approach is then proposed in this study to extensively exploit the design change propagation properties of the feature-driven engineering process. The key of this approach is the introduction of a special set of version annotation attributes

for each configuration version to capture the change propagation semantics. With the help of the version annotations, the configuration versions can be easily constructed without data redundancy and all the constituent object versions with their values can be easily identified and accessed.

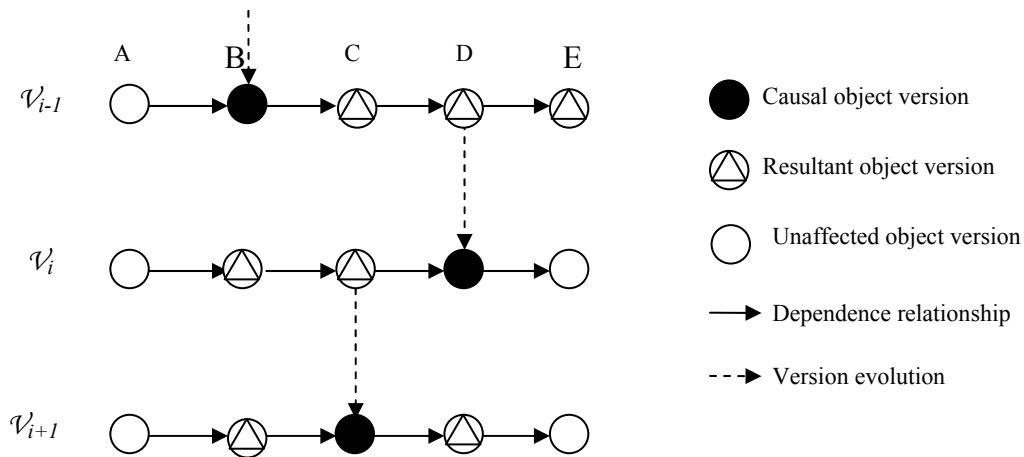


Fig. 5.5 Design change propagation scope and object version identification

In a feature-driven engineering process, the underlying motivation to launch a new configuration version is to react to the design changes made to one of the constituent design objects between its new and old versions. Consider a simple feature-driven engineering process (Fig. 5.5) which contains five tasks to generate five design objects, A , B , C , D and E , respectively. E is dependent on D which is recursively dependent on C till A . While working on a configuration version V_i , the user intends to try an alternative of C and create a new version of C . Immediately after starting the operation to create the new version C , the creation of the new configuration version V_{i+1} containing this new version of C is launched. Creation of new version C is successful only in the case that the configuration version V_{i+1} is correctly “pre-created”. In the course of constructing the configuration version V_{i+1} , the user explicitly declares the new version for object C as the causal version. He then makes a decision on the design

change propagation scope that the evolution of C will cause. In this case, objects B and D should make a resultant design change. He declares the versions of these two objects as resultant versions in the new configuration version. All the design object versions have no independent version numbers. They are uniquely identified by a pair: their object name identifiers and the version number of the configuration version to which they belong. In this way, the interdependent design objects evolve in phase to migrate from one version state to a new version state. All these information about the causal and resultant design object versions in the new configuration version is stored as the value of the annotation attributes, “*Launched-by*” and “*The-set-of-resultant-design-object-versions*” respectively. For the unaffected object versions, no explicit records are given to them, because they can be automatically identified by the information stored in the annotation attributes. Firstly, all other object versions apart from the causal and resultant versions are deemed as unaffected object versions. No “pre-creation” of new versions for these objects is needed and their values are identical to that of the same objects in their parent configurations which are identified by another annotation attribute, “*Is-a-decedent-of*”. For example, the values of objects A and E in configuration version V_{i+1} are identical to the counterparts in configuration version V_i . These value inheritance relationships may be recursive, like the way the object A behaves: its value in configuration version V_i is inherited from a further upper level ancestor in configuration version V_{i-1} . Once an operation needs to retrieve the real value, a simple resolution procedure is called to locate the original object version with a valid value according to its object name, the current configuration version it belongs to and further the value of its attribute “*Is-a-decedent-of*”.

In the example above, the dependence relationships do not need to be explicitly defined and stored in the management database. The laborious construction process is avoided to associate interdependent object versions with a set of references every time to create a new configuration version. Supports to the control of the dependence relationships are implemented in the computation logic layer which co-functions with another sub-system in the entire engineering environment to perform process management. The detail of this sub-system is depicted in next chapter. Simply put, for every configuration version, a dynamic design flow is configured to trace the tasks that have been done, the tasks that are at working and the tasks that are permitted or expected to come out next, using the interdependence knowledge.

To implement this versioning strategy for the control of design change propagation in a feature-driven engineering process, the configuration version (database version) is only required to be instanced with a special identifier from a class type containing dedicatedly defined version annotation attributes. The identifier of the configuration version is bound to a unique version number to identify itself and all its constituent design object versions which have no independent version number. The annotation attributes store adequate information to identify the unaffected object versions which are not explicitly replicated to avoid data redundancy.

5.2.3. Control of Configuration Version Creation

The end-user has full control on the configuration version creation, but in an indirect way. The root configuration version is pre-created when a new project is created. The subsequent configuration versions are created in the interim to create a new design object version. The relationship between the project, configuration version and design

object version is illustrated in Fig. 5.6. According to the design change propagation properties, some design objects are not allowed to proactively undergo design changes because they are always automatically derived from upstream design descriptions. To implement this constraint in the management system, a special attribute, *IsProactive* is incorporated into the generic design object class (Fig. 5.6). The value of this attribute is predefined in the system during system development based on the engineering process knowledge. The inheritance property ensures every instance design object version carries this attribute.

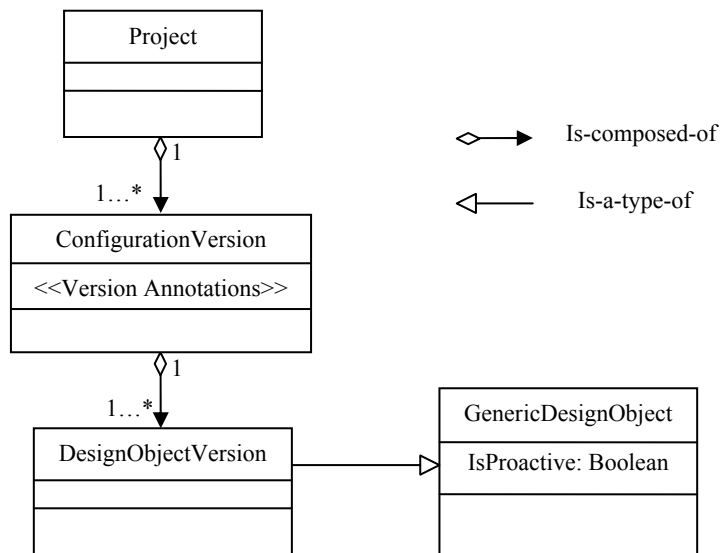


Fig. 5.6. Information structures and the *IsProactive* attribute

Creating a new configuration version (excluding the root configuration version) is performed in the following way (see Fig. 5.7). When the user or an engineering application requests through the wrapper an update to a design object in the shared repository, there are two possibilities. Firstly, if the value of the *IsProactive* attribute is true, there are two options for the end-user to select: either overwriting the previous version, or creating a new version. Further, if he selects overwriting the previous version, no new version for this design object, as well as for the configuration version, is created; if he selects creating a new version, before a new version of this design

object is created, a new configuration version containing this design object version is firstly created. Secondly, if the value of the *IsProactive* attribute is false, creation of a new version for this design object is prohibited and only a direct overwrite operation is permitted.

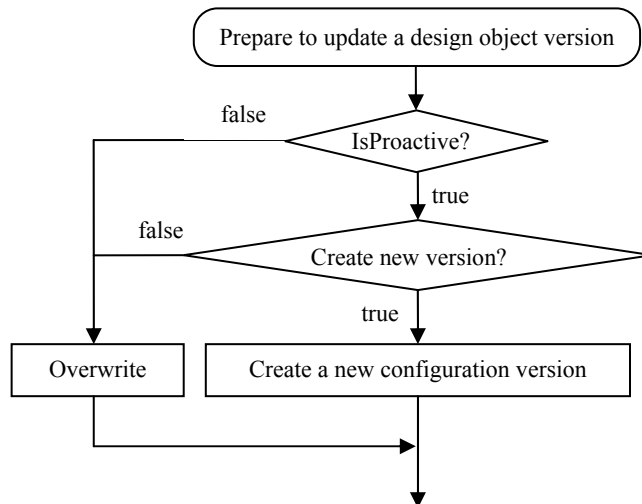


Fig. 5.7 The computation logic to control creation of configuration version

The next section shows a set of operations on the configuration versions and design object versions to support the maintenance of data consistency and integrity with arbitrary design change causes. To emphasize again, as a result of the comprehensive consideration of the design change propagation properties deliberately identified, the implementation to control the design change propagation is quite straightforward and the system developed accordingly is easy to operate.

5.3. Specification of Operations

In this section, the methodological issues about the required version control and configuration management operations on the management database are discussed. Basically, the operations can be grouped into three categories: operations on projects, operations on configurations or configuration versions and operations on design objects or design object versions. They can also be classified into simple or complex

operations. One simple operation contains exactly one query to the database. On the other hand, one complex operation may contain multiple queries. As stated, simple operations on configuration versions are performed by versioning transaction and simple operations on design object versions are performed by non-versioning transaction. Similarly, simple operations on the projects are performed by project transactions. Complex operations may be performed by a group of different types of transactions which are adequately nested.

Table 5.1. Operations on projects, configurations and design objects

Category	List of operations	
	Simple operations	Complex operations
Operations on projects	Open, Close, Delete, Rename	Create, Import, Export
Operations on configurations	Delete, Reconfigure	Make-All-Inclusive
Operations on design objects	Delete, Check-Out	Check-In

5.3.1. Operations on Projects

The *Open*, *Close*, *Delete* and *Rename* operations are all simple and the roles they take are self-explanatory. Other three operations on a project are explained in detail in the following.

- **Create**

A project has a set of attributes and may contain one or many configuration versions (Fig. 5.4). The *Create* operation defines all the attributes of a newly created project. Further, an empty root configuration version or an imported configuration version is initialized in the project container. In the case of initialization with an empty version, the constituent object versions belonging to the configuration version are pre-created with *nil* values. The object versions are then updated in the standard way using the

non-versioning transactions. In the case of initialization with an imported configuration, the pre-created configuration is imported from other projects instead of a newly created one. The *Import* operation on projects is invoked implicitly.

- **Import and Export**

After the initialization of a project, the newly created root configuration version may be explicitly overwritten by the *Import* operation internally or the *Export* operation externally. Specifically, the *Import* operation copies a full configuration version used in another project into the current project to replace the existing root configuration version. Likewise, the *Export* operation copies a full configuration version used in a current project into another project. The configuration version imported or exported should be all-inclusive (see below). Consequently, the *Make-All-Inclusive* operation will be invoked implicitly if unaffected versions with implicit values exist in the configuration version to be imported or exported.

5.3.2. Operations on Configurations

There are only a few types of explicit operations on configurations. This is because some operations on configurations are performed in the complex operations on projects or design objects. For example, the creation of root configuration versions is performed inside the project creation operation and the creation of subsequent configuration versions is performed inside the design object creation operation. Deletion of a configuration version is attributed to the *Delete* operation. Two other operations are explained in detail in the following.

- **Reconfigure**

The *Reconfigure* operation is a simple operation and is used to edit the properties of a configuration version except the root configuration version, especially the annotation attributes. It provides a way to modify the design change propagation scope after the initial definition. In another words, it allows redefinition of the composition of a configuration version in terms of declaration of the causal object versions, resultant object versions and unaffected object versions contained in the configuration version.

- **Make-All-Inclusive**

In a configuration version, only the causal and resultant object versions are explicitly recorded by a physical value which is identified by its object ID and the corresponding configuration version number. The unaffected object versions are actually not explicitly included in the configuration version. Retrieval of the values of these object versions would need a dynamic translation of the relevant implicit information into explicit representations. This is not convenient in some cases when the user wants to browse in between the constituent object versions of a configuration or make group copy/check-out operation. Therefore, it is sometimes desirable to perform a collection of translation operations on all unaffected object versions in one turn to make them all explicitly and permanently represented by a static valid value like the casual and resultant object versions. The *Make-All-Inclusive* operation is responsible for this function. Upon executing this operation, the normal dynamic translation process is no longer required when retrieving the unaffected object versions next time. The *Make-All-Inclusive* operation is a complex operation, since it may recursively execute a set of versioning and non-versioning transactions.

5.3.3. Operations on Design Objects

Operations on design objects support the design transaction model depicted in Chapter 3 and the augmented layered transaction schema depicted in section 5.2.1. It may involve operations on the configuration version the corresponding design object version belongs to. This has been explained in section 5.2. In brief, three compact operations are used by the end user to explicitly operate on the design object versions while the configuration versions may be affected meanwhile.

- **Delete**

To delete an object in a particular configuration version, it is sufficient to update it with the *nil* value.

- **Check-Out**

To physically check-out a design object version belonging to a configuration version, its value must be identified and retrieved. This is presented in section 5.2.2. The actual *Check-Out* operation may be required to be executed within a group, and Check-out of this group of design objects bring on a set of virtual Check-In operations and virtual Check-Out operations (see section 3.4.3).

- **Check-In**

The check-in operation is one of the most important operations to implement the proposed versioning and configuration scheme. It can only be allowed when the *IsProactive* attribute is true. By a successful *Check-In* operation, the modified design object may be returned to create a new version or simply overwrite its former value. This has been presented in section 5.2.3.

5.4. Application of the Proposed Model in the Integrated Progressive Die Design and Manufacturing Engineering Environment

This section shows how the proposed version control and configuration management model can be applied to the progressive die design and manufacturing processes to offer desirable version control and design change propagation management assistance. The composition template of a progressive die design and manufacturing project has been shown in Fig. 5.1. All the corresponding dependence relationships involved have been illustrated in Fig. 3.7. A sample versioning scenario is used to elaborate how the desirable versioning control and design change propagation support is achieved on a computer-based platform via performing corresponding operations defined above. In this scenario, the basic product design and manufacturing solution (Con_1) is expected to spawn three tentative alternatives (Con_2 , Con_3 and Con_4 respectively) corresponding to three original design changes made to three different constituent design objects in Con_1 . Fig. 5.8 shows the configuration version derivation graph which has three version branches corresponding to three versioning steps which are detailed in the following. Thorough understanding of this case study needs some progressive die design and manufacturing process knowledge which can be found in references (Cheok & Nee 1998a, b; Jiang *et al.*, 2000; Zhang *et al.*, 2002; Cheok 1998; Lee *et al.*, 1993.)

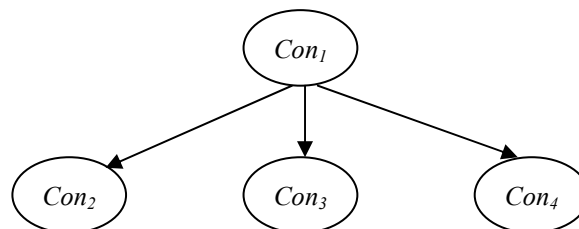
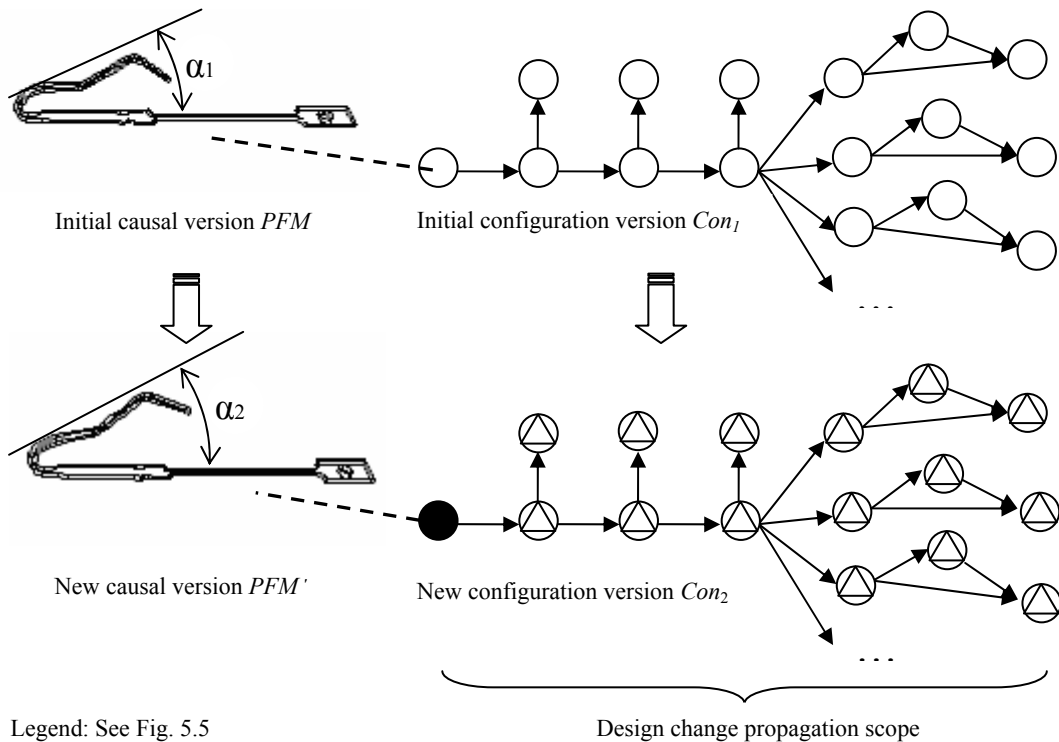


Fig. 5.8 The configuration VDG for the example scenario

- **Versioning step 1: propagation of a design change to generate Con_2 from Con_1**

This is a slight change made to the product design or the *Product-Feature-Model* (*PFM*). Specifically, angle α_1 is changed to α_2 , as shown in Fig. 5.9. This will prompt a new version of *PFM* and further, a new configuration version coexistent with the old ones. Suppose the initial *Product-Feature-Model* version is *PFM*. The check-in of new version *PFM'* causes change propagation. New configuration version *Con₂* is spawned to incorporate *PFM'*. Since the change on the product definition will affect all other design objects, the design change propagation scope in this case expands the whole configuration and there is no unaffected version. The attribute value of the “pre-created” *Con₂* will be: “*Is-a-descendent-of*” = *Con₁*; “*Launched-by*” = *PFM*; “*The-set-of-resultant-design-object-versions*” = {*FP, FPD, DO, SL,*}. The initial value of the “pre-created” design object versions in *Con₂* apart from *PFM'* are all *nil*. After *Con₂* is successfully “pre-created”, a new dynamic design flow is generated for this configuration version to assist the user to trace the check-out of the tasks to be done and check-in of them when they have been done through non-versioning transactions. The detailed operation sequence for this versioning step is shown as follows:

- ① *Check-in a PFM*
- ② *Update / Create a new version? – Create a new version.*
- ③ *Create a new configuration version, initialize its annotation attributes and the constituent design objects with nil value or imported values*
- ④ *Select a design object from the dynamic design flow browser for Check-out*
- ⑤ *Work on the design object in the work-space*
- ⑥ *Check-in the work-done design object for update*
- ⑦ *Repeat steps ④-⑥*



Legend: See Fig. 5.5

Fig. 5.9. Step 1 in the scenario: generating Con_2 launched by a design change on *Product-Feature-Model*

- **Versioning step 2: propagation of a design change to generate Con_3 from Con_1**

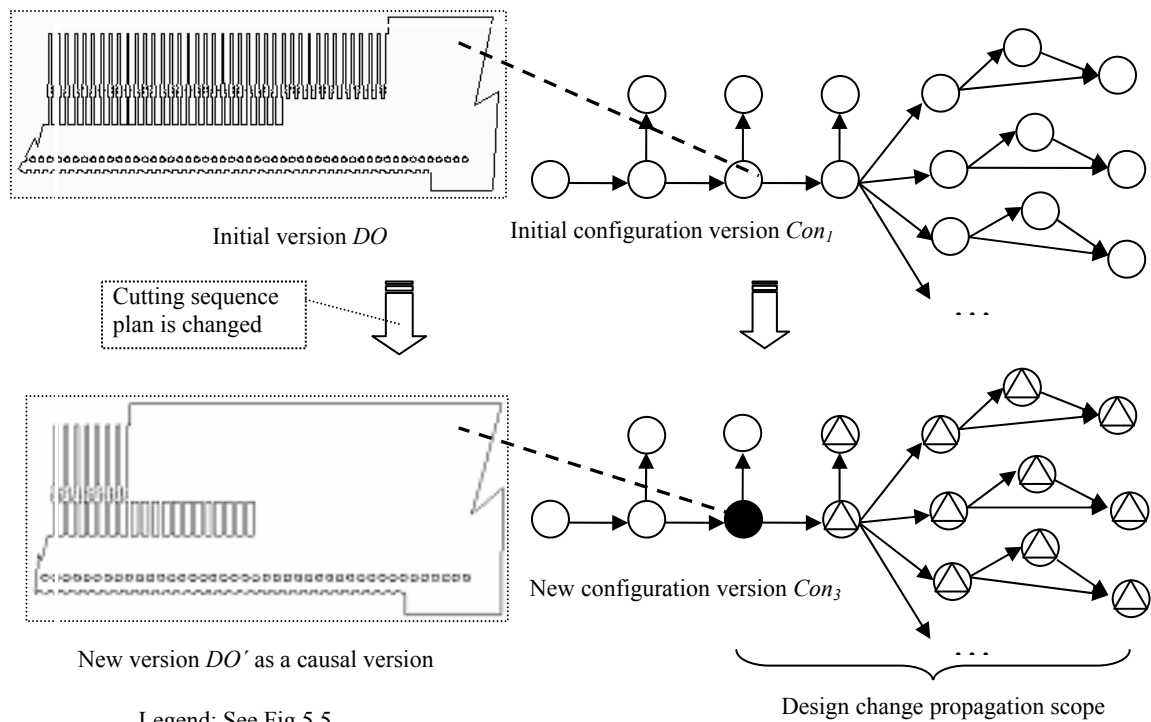
This is a design change made to *Die-Operation-Feature-Model (DO)*. Under normal circumstances, the particular iterative process to generate *Die-Operation-Feature-Model (DO)* from *Flat-Pattern-Feature-Model (FP)* only involves the stamping process planning or strip layout (nesting and staging) without consideration of the placement of the punch on punch plate. The resulted *DO* is only a guess and tentative for confirmation by incorporating constraints which become explicit till to the next design stage. This will cause many tentative alternatives, potentially regarded as versions, most of which are only valid within the current design stage. In an integrated intelligent die design system like the IPD system (Jiang *et al.*, 2000), the strip layout design process also includes the design of the shape of the upper body of the punch by adding some additional information such as insert parameters, relieves, *etc.*, into the die operation feature model (see also section 3.2.1). This makes localized punch

contour constraints when staging the stamping operations. The version explosion problem is thus avoided by the users' local iteration to eliminate unaccepted solutions immediately. If the use or disuse of an alternative still cannot be decided even having considered the contour constraints, it is a real case to embrace two or more versions of *DO* with respect to the entire configuration. In this case, only after almost all design objects consistent with an alternative in the entire configuration are generated, can the user make the last decision to to use or disuse this alternative. Fig. 5.10 shows the way to explore such kind of alternative corresponding to a design change adopting a new piercing sequence plan and thus updating the value of the *Die-Operation-Feature Model* version from *DO* to *DO'*. The check-in of *DO'* causes change propagation. New configuration version *Con₃* is spawned to incorporate *DO'*. Since the change on *DO* will only affect all downwards design objects and not the upwards design objects, the design change propagation scope in this case covers all the design objects directly or indirectly dependent on *DO*. The attribute value of the "pre-created" *Con₃* will be: "Is-a-descendent-of" = *Con₁*; "Launched-by" = *DO*; "The-set-of-resultant-design-object-versions" = {*DC, DB, DW, P₁FM, P₂FM, P₃FM, P₄FM,*}. All design object versions other than the causal and the resultant ones are unaffected versions. After *Con₃* is successfully "pre-created", a new dynamic design flow is generated for this configuration version to trace the check-out of the tasks to be done and check-in of them when they are done through non-versioning transactions. The detailed operation sequence for this versioning step is shown as follows:

- ① *Check-in a DO*
- ② *Update / Create a new version? – Create a new version.*
- ③ *Create a new configuration version, initialize its annotation attributes and the constituent design objects with nil value or imported values*
- ④ *Select a design object from the dynamic design flow browser for Check-out*
- ⑤ *Work on the design object in the work-space*

⑥ Check-in the work-done design object for update

⑦ Repeat steps ④-⑥



Legend: See Fig.5.5

Fig. 5.10. Step 2 in the scenario: generating Con_3 launched by a design change on *Die-Operation-Feature-Model*

- **Versioning step 3: propagation of a design change to generate Con_4 from Con_1**

This is a design change made to the shape of a notching punch on the strip layout to help save the costs of making the die. It so happens that this modification will affect the external profile of the flat pattern and hence the actual product. A new version of this notching punch feature model and a further new configuration version consisting of a corresponding new product design will be generated so that the die designer can discuss the effect of these changes with the product designer. Suppose the initial feature model for the notching punch is P_4FM (*Part4-Feature-Model*). The designer tries to widen the slat of the punch so that the corresponding wing of the stamped-product is shrunken, *i.e.*, distance d_1 is changed to d_2 (Fig. 5.11). Since P_4FM is automatically derived from its upstream design description, the *Die-Configuration-*

Feature-Model (DC), and further the *Die-Operation-Feature-Model (DO)*, it is not allowed to proactively perform design changes. Creation of new version of P_4FM does not begin from a check-in operation on P_4FM as in step 1 or step 2, but in an indirect way through a modification to DO and then DC , which then generates the desired new P_4FM version P_4FM' . The check-in of the modified DO' causes change propagation. New configuration version Con_4 is spawned to incorporate DO' , DC' and P_4FM' . The change on DO in this case will affect all upwards design objects as well as partial downwards design objects: most of the relevant part feature models (except P_4FM and some plates in the die structure), as well as the corresponding part process plans and part NC code documents may not be affected by this design change. Therefore, the design change propagation scope in this case covers all the upwards design objects and partial downwards design objects. The attribute value of the “pre-created” Con_4 will be: “*Is-a-descendent-of*” = Con_1 ; “*Launched-by*” = DO ; “*The-set-of-resultant-design-object-versions*” = $\{PFM, FP, FPD, DO, SL, DB, DW, P_1FM, P_4FM, \dots\}$. All design object versions other than the causal and the resultant ones such as P_2FM , P_3FM , are unaffected versions. After Con_4 is successfully “pre-created”, a new dynamic design flow is generated for this configuration version to trace the check-out of the tasks to be done and check-in of them when they are done through non-versioning transactions. The detailed operation sequence for this versioning step is shown as follows:

- (Decide to try a modification to a notching punch P_4FM)
- ① Check-out Die-Configuration-Feature-Model DO
 - ② Edit DO to DO' so that the desired P_4FM' can be achieved
 - ③ Check-in DO'
 - ④ Update / Create a new version? – Create a new version.
 - ⑤ Create a new configuration version, initialize its annotation attributes and the constituent design objects with nil value or imported values
 - ⑥ Automatically generate DC' , and further P_4FM' , P_1FM' , P_3FM' , etc. from DO' to replace DC , P_4FM , P_1FM , P_3FM , etc.

- ⑦ Select a design object from the dynamic design flow browser for Check-out
- ⑧ Work on the design object in the work-space
- ⑨ Check-in the work-done design object for update
- ⑩ Repeat steps ⑦-⑨

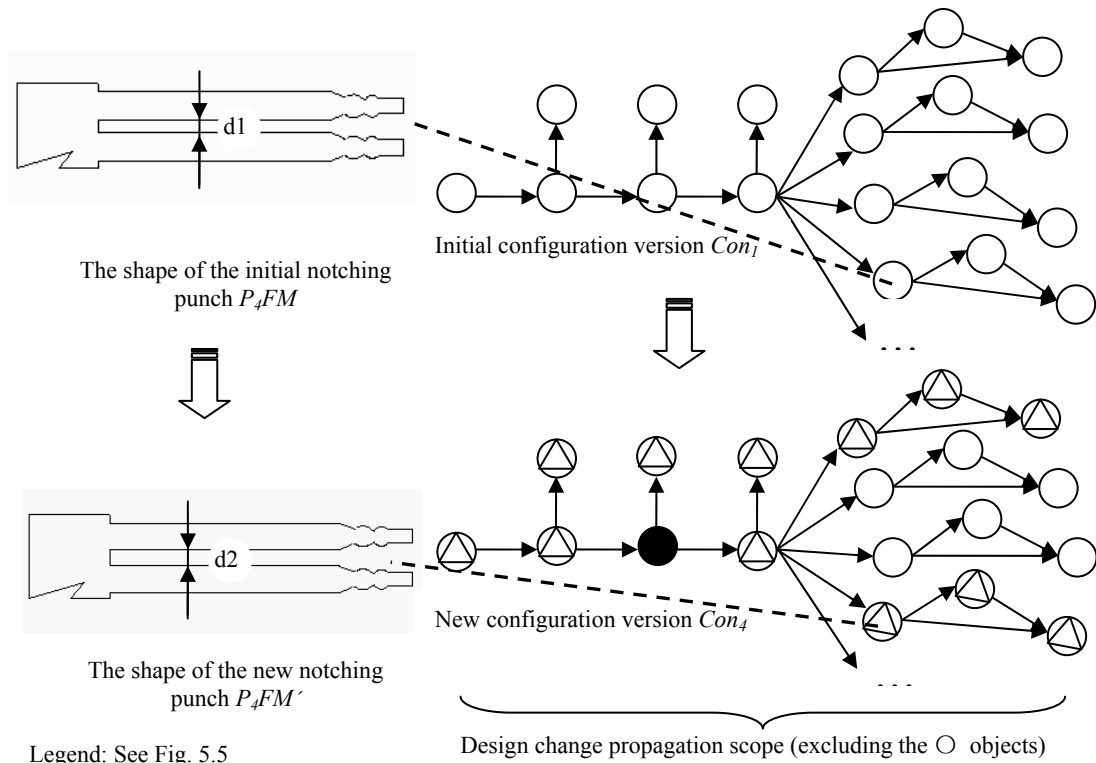


Fig. 5.11. Step 3 in the scenario: generating Con_4 launched by a design change on $Part_4$ - $Feature-Model$

• Summary

The above versioning scenario consisting of three steps is used to demonstrate the normal operations involved. In real circumstances, some other operations discussed in the previous section, such as “Reconfigure”, “Delete”, *etc.*, may also be involved. It should be pointed out that the proposed version control and configuration model has attempted to be considered as theoretically comprehensive as possible. It can cover all possible cases which may occur in the progressive die design and manufacturing process. Furthermore, not only is the mechanism applicable to the progressive die design and manufacturing process, but it can also be geared to other types of feature-driven processes, such as the integrated product and mould design and manufacturing processes, the integrated product and fixture design and manufacturing processes, *etc.*

5.5. Towards a Comprehensive Information Model and a Full-fledged GUI Design

Section 5.2.3 has shown the information structure for management of configuration and versioning with the emphasis to reveal the relationships between the project, configuration and design object. Clearly, this information structure only reflects a small part of the information requirements for the whole CAX framework-based network-integrated engineering environment. To derive a comprehensive data schema, other important issues that need to be considered include logical distribution of design data and design activities, design transactions and run-time information management, engineering process management, *etc.* The main method used to derive this data schema contains the components of perception, representation and validation while the modeling process is incremental. Different aspects of the engineering environment are expected to be specially addressed and represented in the overall data schema through a procedure of gradual refinement. To make the presentation concise, the comprehensive information model is only presented once in the next chapter after the process management issue is addressed.

Similarly, a full-fledged GUI design is also only presented once in the next chapter after the process management operations become clear like the configuration management and versioning control operations depicted above.

CHAPTER 6

ENGINEERING PROCESS MANAGEMENT

Based on process decomposition, analysis and re-engineering, Chapter 3 has presented a first order approximation of the engineering process representation, a IDEF0 activity model. However, the IDEF0 activity model is only a static snapshot of the engineering process and further efforts are needed to develop an implemental model which can be incorporated into the CAX framework to provide process management support. This chapter presents a process management mechanism and addresses multiple process management issues especially in the aspect of process modelling. A comprehensive data schema is derived while taking into account the information requirements for process management, as well as other functions such as configuration and versioning management presented in the previous chapter. Validation of the data schema and the system behaviour is carried out through examining the sequence diagram for a typical use case. A full-fledged GUI is designed and some experimental results working on the prototype system through this GUI are reported to demonstrate the effectiveness of the proposed CAX framework-based integration approach.

6.1. A Process Management Mechanism Based on Design Flow Configuration

A network-integrated engineering environment is expected to be much more than the incorporation of a platform on top of which a couple of tools can be tied together. On the other hand, many integration-flavored functions shared by the participating tools can be available to assist the end-users. Examples of such functions are product data management (data-centric integration) and process management (process-centric integration). Up to the previous chapter, only product data management functions are addressed without exceeding the concerns of the traditional data-centric integration approach. The limitation of the data-centric integration approach is that the end-users have to assume full responsibility for work-flow control, data consistency, integrity, maintenance and inter-process coordination and cooperation, generally without computer augmentation (Jeng & Eastman 1999). This section presents an advanced integration approach which is both data- and process-centric and allows an expanding set of system capabilities that off-load engineers from some of the above responsibilities and complexity.

6.1.1. Overview

If described in terms of a state-space model, the behaviour of a computer-supported engineering process can be represented by the following equations in terms of discrete functions:

$$\mathbf{x}(n+1) = \mathbf{f}(\mathbf{x}(n), \mathbf{u}(n)) \quad (6.1)$$

$$\mathbf{y}(n) = \mathbf{h}(\mathbf{x}(n)) \quad (6.2)$$

where $\mathbf{x}(n+1)$ and $\mathbf{x}(n)$ denote the global design state at time $n+1$ and n , $\mathbf{u}(n)$ denotes design operator through the user interface, $\mathbf{y}(n)$ denotes the external appearance of the design state in the user interface window.

The state-space model dictates that an integrated engineering environment may assist the end-user in four aspects:

- Design tracking: keeps track of the state of the design and the design history—maintain the state sequence $\mathbf{x}(n)$ in the integration infrastructure. All the tasks involved in the activity model are no longer treated individually and in an isolated way. Instead, a full picture about the progress of the whole project is captured in a certain way to make explicit what needs to be done throughout the process and what has been done so far. The engineering tools integrated into the infrastructure, as well as the end-users, may be informed about or control the migration of the design state.
- Design state browse: provides facilities that allow the design engineer to browse in a highly convenient way through the administered state of design—realize function $\mathbf{h}(\cdot)$ to obtain a virtualized output $\mathbf{y}(n)$. Not only is the design state informed to the end-users, but also in an intuitive way.
- Process execution guidance: supports the design engineer in efficiently executing design activities—provide implications of the desired $\mathbf{u}(n)$ at any moment during the process execution course. This assistance is probably mixed with the second aspect. Given that the integration infrastructure is “aware” of the state of design and is aware of possible ways of transforming this state of design to a new state, then it can advise the design engineer on tasks to perform next.

- Constraint enforcement: permits constraints on the design process to be defined and enforced—exploit the process knowledge related to state transformation function $f(\cdot)$ to assist the end-users. At any moment in between the starting and the ending point of an engineering process, a very knowledgeable integration system will allow only runs of tools for which valid input data is present, and support the design engineer by indicating which tools can and should be run. This helps to make the design process less error-prone and to improve productivity.

All these potential engineering process management functions can be realized in the CAX framework-based process integration environment (Fig. 4.4). The mechanism is related to a set of interrelated techniques around a concept of design flow configuration. A design flow is a description of a design process in terms of design activities and temporal data, and control dependencies between design activities. In the CAX framework kernel, both the design flow configuration information and its run time information are maintained. The configuration information is defined based on the process knowledge, one possible description form of which is the IDEF0 activity model. As a template defining placeholders for actual data, the design flow configuration is defined before the actual engineering process is started and relatively stable. The run time information is updated continuously in the course of the engineering process. It “colours” the template design flow by filling the placeholders with actual data items consumed and produced during actual tool runs.

There are three ways to define a design flow configuration. In the case of well-structured processes, the design flow is both predicted and repeated and can be described precisely. Therefore, the corresponding design flow configuration logic can

be predefined in the framework either within the kernel or the workbench application. No operations on the design flow configuration are opened to the end-users. In the case of unstructured processes, they are executed in a spontaneous and rather *ad hoc* manner (often called *ad hoc* workflows in the literature on workflow management (Dellen *et al.*, 1997)). Comprehensive facilities need to be provided by the framework to the end-users to configure a design flow from scratch using primitives defined by a higher level meta process model or design flow templates. Located in the middle of the spectrum is the semi-structured process. Some parts can be precisely described by process fragments while others are determined by the creativity of the end-users. In the example case of the progressive die design and manufacturing process, it is almost a completely structured process apart from the design flow configuration of the CAPP (Computer-Aided Process Planning) tasks. The tasks for every progressive die part in need of performing CAPP are identical, but the number and the specific parts with an identifier cannot be pre-determined in advance until the die configuration task is finished. One possible way is to directly acquire this information through access to the internal of the feature-based progressive die description model. However, the framework is only allowed to access the meta data and this direct way should be avoided. Therefore, while a basic design flow can be presented to the end-users at the start of the process, it needs to be refined dynamically by small interventions. In the simplest case, the interventions can be done manually by the end-users through an interface within the workbench application. In the complex case, it can be done automatically by the workbench application which can identify those die parts in need of performing CAPP. The mechanism is like this: After the die configuration task is performed, a feature-based description for every die plate and punch is generated and stored in the shared repository in the form of a file. The meta data information

corresponding to these files is exactly the required information to configure the design flow for CAPP tasks. Therefore, the workbench application can query the meta data database to acquire the required information to refine the design flow configuration dynamically. While the structured and semi-structured process can be defined with the exactly required resources towards a compact system, they can also be viewed as an unstructured process towards a more generic and flexible system.

With the design flow configuration and run time information maintained in the framework kernel, the domain neutral workbench application provides facilities to virtualize the design flow in a flow browser. The virtualization method used by the flow browser is termed flow colouring (ten Bosch *et al.*, 1993): presenting the intricate relationships among tools and design objects in an intuitive way and further tracking the activities generating these design objects in the context of four possible lifecycle states respectively: “not ready”, “ready”, “done”, and “active” (see next sections). The flow browser permits the end-user to interact with a coloured design flow. It presents information about the structure and status of the design in an attractive and comprehensive way to the end-user. It also offers convenient means to navigate through the available information, to explore the state of design. That is, it makes the advanced process management services available to the end-users.

6.1.2. Process Representation

The first step to implement the above process management mechanism into the CAX framework-based network-integrated engineering environment is to develop an appropriate process representation which can be integrated into the management database data schema. At the highest level of abstraction, an engineering process is a

design flow which may consist of sub-flows. Further, a design flow must be modelled to offer constructs to describe engineering activities and dependencies between them. In generic sense, the design flow model for a feature-driven engineering process with three possible types of tools (*i.e.*, automatic, semi-automatic and manual, see Chapter 3) involved can be derived from a meta process model shown in Fig. 6.1. This model is read as in the following. *Design Flow* is a general term for referring to a representation for the overall process or any levels of sub-process. It has a recursive definition and the leaf construct is *Activity*, *i.e.*, the *Activity IS-A Design Flow*. The links between design activities and/or sub-flows are all captured in the *Precedence Relationship* class which is a tuple of *Current (design) Flow* and *Preceding (design) Flow*. The *Activity* has consumed and produced *Design Objects*. Its “state” attribute gets the following values: {“not ready”, “ready”, “done”, and “active”} which are somewhat self-explanatory.

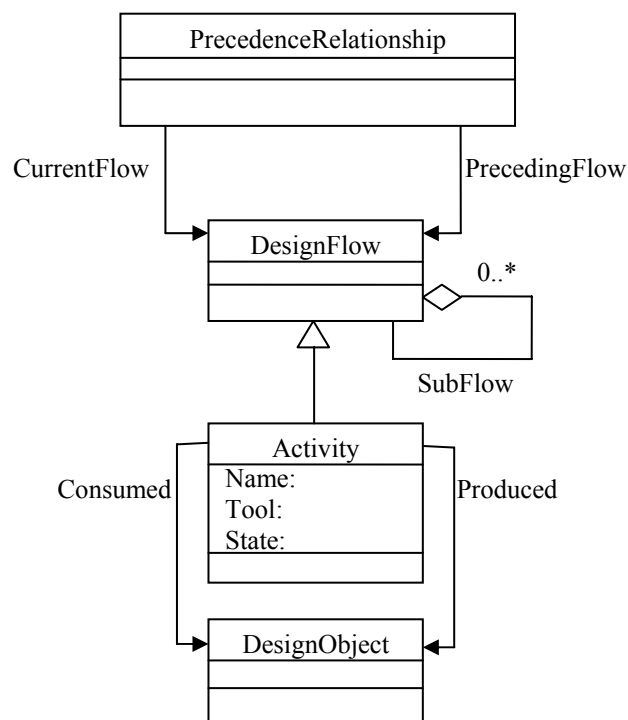


Fig. 6.1. A meta process model for feature-driven engineering process

The key constructs of the above generic design flow model are illustrated by the example compound design flow *F1* in Fig.6.1. *F1* has a sub-flow *F2*. Activity *A1* and *A2* are *F1*'s subtype instances. Activity *A3* and *A4* are *F2*'s subtype instances. At the sub-flow level, *A1* is before *A2* which is before *F2*. This is represented by *PR1* and *PR2*. At the activity level, *A2* is further refined with precedence both before *A3* and *A4* while *A3* and *A4* can be concurrent. This is represented by *PR3* and *PR4*. With appropriate methods defined in *Design Flow* class, definition of *F1* can be transmitted from sub-flow level (diagram (b)) to activity level (diagram(c)) dynamically. In other words, the compound design flow *F1* can either be in the form of (b) or (C). This simple example is helpful for understanding the nature of the progressive die design and manufacturing process which includes a CAPP sub-process that is unable to be decomposed to the activity level until certain information is available.

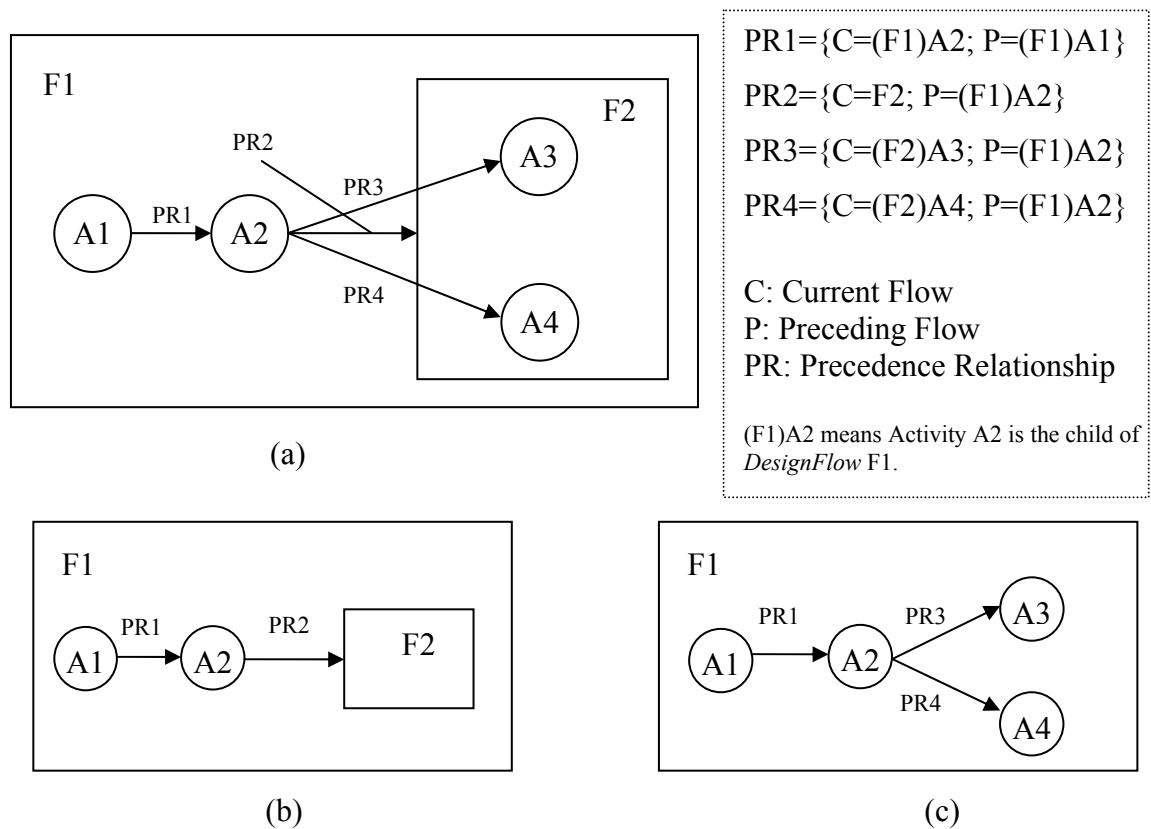


Fig. 6.2. Example compound design flow containing two activities and a sub-flow

6.1.3. The Process Execution Engine

The next essential step to implement the process management mechanism is to define the process execution engine. Basically, process execution involves traversing the activity precedence graph subject to: (a) precedence relationships among activities; (b) user actions; (c) resource availability (*i.e.*, tool, network-connection availability). The execution algorithm checks constraints at two levels:

- First level: State transition for each activity involved. For example, activity state is transited from “not ready” to “ready” when its predecessors are complete.
- Second level: State transition for the process or sub-processes. For example, a sub-process can be transited from “black-box” state to “white box” state when this sub-process can be recursively decomposed to activity level so that the user can be instantly informed with the tasks to be done in detail through the design flow browser.

Physically, the process execution engine is located in the CAX framework kernel, or specifically, the Data and Process Management Kernel (Fig. 4.1). It monitors the user actions and the engineering tools’ interactions with the framework kernel, and makes the corresponding modifications on the run-time representation of design flow configuration persistently stored in the meta data database. Definition of the process execution engine is closely pertinent to the ways to define the design flow configuration which carries the semantics of the activity precedence relationships. From the viewpoint of process execution engine, the three process definition ways viewing a process as unstructured, structured and semi-structured respectively, both have advantages and drawbacks.

If viewed as unstructured, the design flow configuration is defined through a process configuration engine before it is “executed”. After definition, the design flow configuration information is stored in the meta data database and will be loaded into the process execution engine at run time. It is possible for the already-defined design flow configuration to be redefined or adapted after it has been executed for a while according to the current process status. Therefore, a real-world engineering process may alternatively experience a process definition turn and a process execution turn. One advantage of this way is that the CAX framework can be adaptively configured for different application contexts once it is developed. Another advantage is its self-adaptability in the run-time. The drawback of this way is its system complexity which requires more system development effort.

If viewed as structured, the design flow configuration is hard-coded into the process execution engine. This way sacrifices flexibility and adaptability of the developed system, but saves system development efforts. Representing the dependence relationships into the data schema is unnecessary because the execution engine has this knowledge. Only isolated *Activities* need to be modeled in the data schema and the execution engine can automatically determine the precedence relationship between two activities according to their identifiers. The process execution algorithm is only relevant to the constraints at the first level. Process modeling becomes simplified, and so does the system implementation. Of course, the prerequisite to use this way is that the process itself is structured.

For the semi-structured progressive die design and manufacturing process focused in this study, some compromises were made between the above two ends. A full-featured process configuration engine is unnecessary and the process execution engine is equipped with almost all relevant process knowledge. Further, a special class is defined in the data schema at the sub-design flow level to make the process execution engine have some limited ability to reconfigure the design flow during the process execution course. The details of this aspect are presented in next section in which a comprehensive data schema is derived from the information requirements both for process management and other relevant functions addressed in the previous chapters.

6.2 A Comprehensive Information Model

According to the generic meta process model shown in Fig. 6.1, the definition of an engineering process involves an information entity of Design Object which is also encapsulated in the product data management model (Fig. 5.6, Chapter 5). According to Chapter 3, there should be a few common information entities representing run-time information for the CAX framework to maintain meta data—design data consistency, so that advanced product data management and process management services can be provided. It is therefore probably the right time at this moment to wrap-up all these dispersed information requirements together to derive a comprehensive meta data schema while refining the meta process model to a specific one for the progressive die design and manufacturing process. With this model, not only the process management mechanism, but also the version control and configuration management mechanism as well, is further revealed and validated from a global information structure view.

Fig. 6.3 shows the developed model with UML (Unified Modeling Language) notation (Fowler & Scott 1997). This model also goes beyond the meta data schema layer and further embraces the framework kernel application classes which operate on the meta data. Specifically, white classes in Fig. 6.3 are framework kernel application classes which are transient; that is, they are internal to the application's memory. Shaded classes represent the meta data schema and are persistence capable; that is, instances of them are stored in the ObjectStore[®] database.

The collection of the transient classes and their relationships is a refinement of the component architecture (Fig. 4.2 in Chapter 4). The “*ProjectManagerServer*” class implements the behavior of a remote “*ProjectManager*” interface and runs on the server as a remote service. The remote interface is also implemented by a class running on the client as a proxy for the remote service. The “*ProjectManagerClient*” makes method calls on the proxy object. RMI sends this request to the remote JVM, and forwards it to the implementation. Any return values provided by the implementation are sent back to the proxy and then to the client's program. Functionally, the “*ProjectManagerServer*” controls user access, authentication, session management, and access to the meta data in the database. It has a number of methods, each of which executes a meta data transaction in the ObjectStore[®] DBMS. The “*ProjectManagerClient*” class corresponds to a daemon in the client machine and makes the framework services available for workbench GUI and CAX tool wrappers. It correctly sequences the meta data operations and design data operations, the latter of which are performed by the “*DesignDataHandler*” class also on the client side.

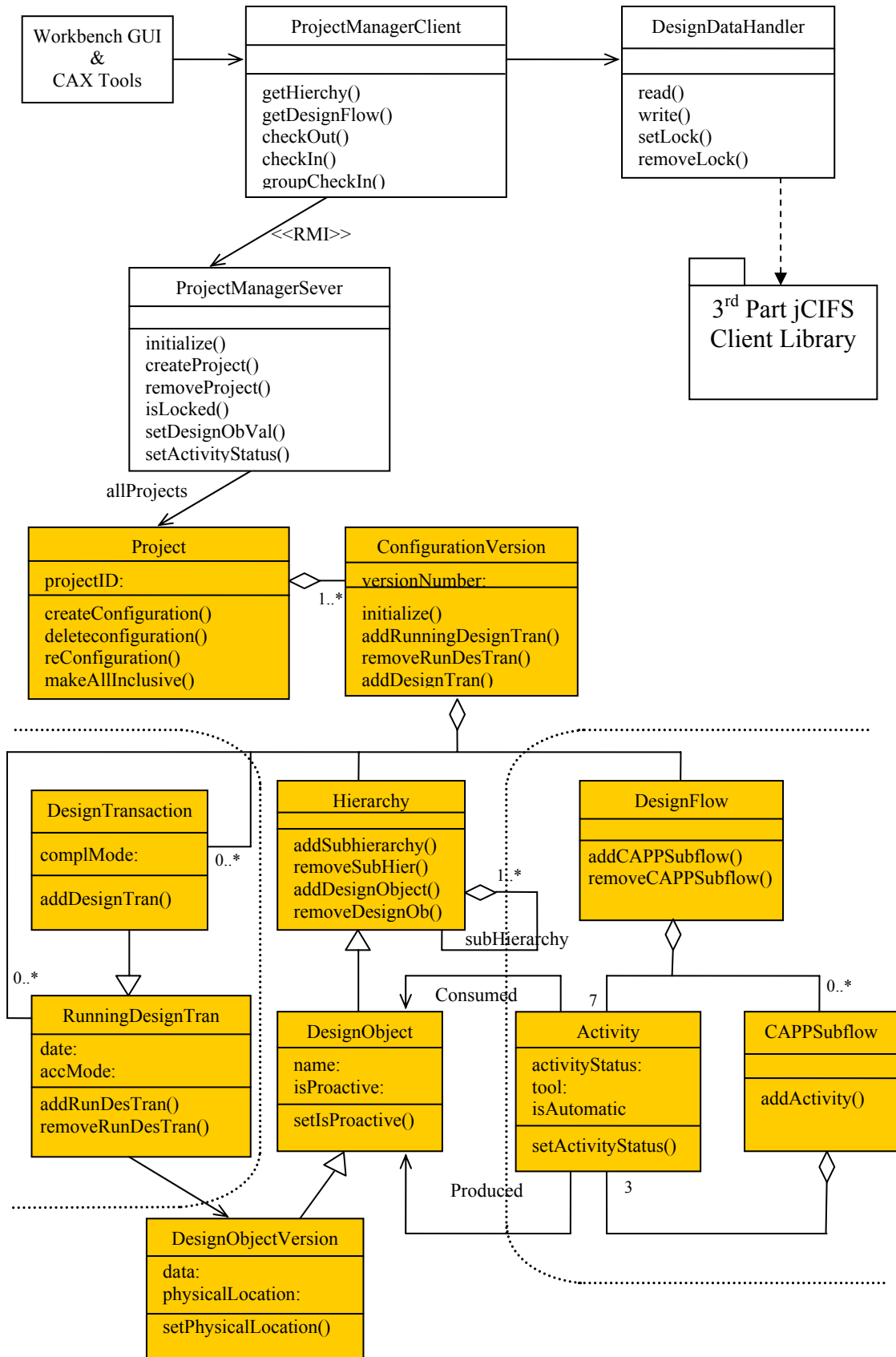


Fig. 6.3. A comprehensive information model for the example implementation

Two dotted lines in Fig. 6.3 divide the shaded meta data schema into three main parts, the run-time part in the left, the product data management part in the middle and the design flow management part on the right.

The run-time part consists of two classes and defines the structure of the run-time information that is maintained to keep track of the design transactions performed to change the state of design. The “*RunningDesignTran*” class defines the running design transaction in which the corresponding design object is checked-out but not checked-in yet. A “lock” is thus applied on this design object in the shared repository to prevent unmanaged overwrites. After the running design transaction is successfully committed with a check-in operation, the corresponding “*RunningDesignTran*” object is destructed, the lock is removed, and another object instantiated from the “*DesignTransaction*” class is constructed in the database to record this committed design transaction. A virtual check-in has the same effect as a physical check-in in this course. The “*DesignTransaction*” class extends the “*RunningDesignTran*” class with an additional attribute “*complMode*” to indicate whether the design transaction is successful or failed. Failed design transaction does not destruct the corresponding “*RunningDesignTran*” object. The “*accMode*” attribute in “*RunningDesignTran*” class indicates whether the access mode is *read-write* or *read-only*. If it is *read-only*, successful check-out of the design object will immediately trigger a corresponding virtual check-in, and drive this running design transaction to be successfully committed. The design transaction group handled at the higher operation level is finally decomposed into individual design transactions represented by these two class objects. With the help of the run-time part information structure, the CAX framework

can maintain the consistency between the meta data and design data. This is the basis for support of product data management and process management.

The product data management part defines the structure of the information that is maintained by the data management services. The central object type is “*DesignObject*”. As a refinement of the information structure shown in Fig. 5.6, all the versioning control and configuration management semantics presented in Chapter 5 is supported. Exceptionally, an object type of “*Hierarchy*” is highlighted in the middle between the “*ConfigurationVersion*” class and the “*DesignObject*” class to further depicts the configuration management semantics. A hierarchy may have multiple sub-hierarchies and the leaf in the hierarchy is design objects. With the help of this “*Hierarchy*” class, design objects in a configuration are organized into a hierarchy like what is shown in Fig. 5.1. The operations on the “*Hierarchy*” object are equivalent to a sequence of bindings by which the composite object refers to its constituents (Carnduff & Goonetillake 2004). Due to the feature-driven engineering process being well-structured, the “binding” process can be designed very easily. Firstly, it is applied on generic design objects rather than individual design object versions, so it is of dynamic binding and can be performed automatically. Secondly, only in two occasions are the Hierarchy generation operations required. The first occasion occurs when a new “*ConfigurationVersion*” object is created and almost all the constituents shown in Fig. 5.1 in the hierarchy are determinate apart from the CAPP group. This is because how many parts with corresponding IDs in the die structure need to perform CAPP tasks is still unknown. The “*initialize()*” method in “*ConfigurationVersion*” class is responsible for this operation. The second occasion occurs when the above information is available and the internal structure for the CAPP group is then generated. The

“*addSubHierarchy()*” and the “*addDesignOb()*” method will be invoked for this operation. After the hierarchy is created, the design object versions in the hierarchy are firstly “pre-created” with *nil* value and then updated with physical value but the “Hierarchy” object itself is relatively static.

The design flow management part defines the structure of the design flow information and the corresponding run-time information that is maintained by the process management services. The central object type is “*Activity*”. As a refinement of the information structure shown in Fig. 6.1, the generic process management semantics presented above is exactly supported while the classes in the current information structure are re-defined in lower abstract level. This is because the properties of the example progressive die design and manufacturing process have been incorporated into the refined model to make the information structure more specific. Each “*DesignFlow*” object is now concretely defined to belong to a “*ConfigurationVersion*” object and no recursive representation for its hierarchy is applied. On the other hand, a design flow is fixedly defined as a two-level composition, *i.e.*, a design flow has seven activities and zero to n CAPP sub-flows, each of which has three further activities. No precedence relationship between activities or sub-flows is captured in this model, because the user applications have been designed to be equipped with such knowledge and can decide the precedence relationships between two relevant activities as long as they are identified. Similar to the product data management part, only in two occasions are the operations on the design flow configuration required. The first occasion occurs when a new “*ConfigurationVersion*” object is created and all the seven constituent activities in the first level are defined with the containers for CAPP sub-flows empty because how many parts with corresponding IDs are in the die structure is still

unknown. The “initialize()” method in “ConfigurationVersion” class is responsible for this operation. The virtualized design flow after this initialization is shown in Fig. 6.4 (a). The second occasion occurs when the above information is available and all the CAPP sub-flows are then defined with three activities in each. The “addCAPPSubflow()” method is invoked for this definition operation. The initial design flow represented by Fig. 6.4(a) is then changed to a new one represented by Fig. 6.4(b). After the design flow is configured, “execution” of the process will update the “activityStatus” attribute accordingly.

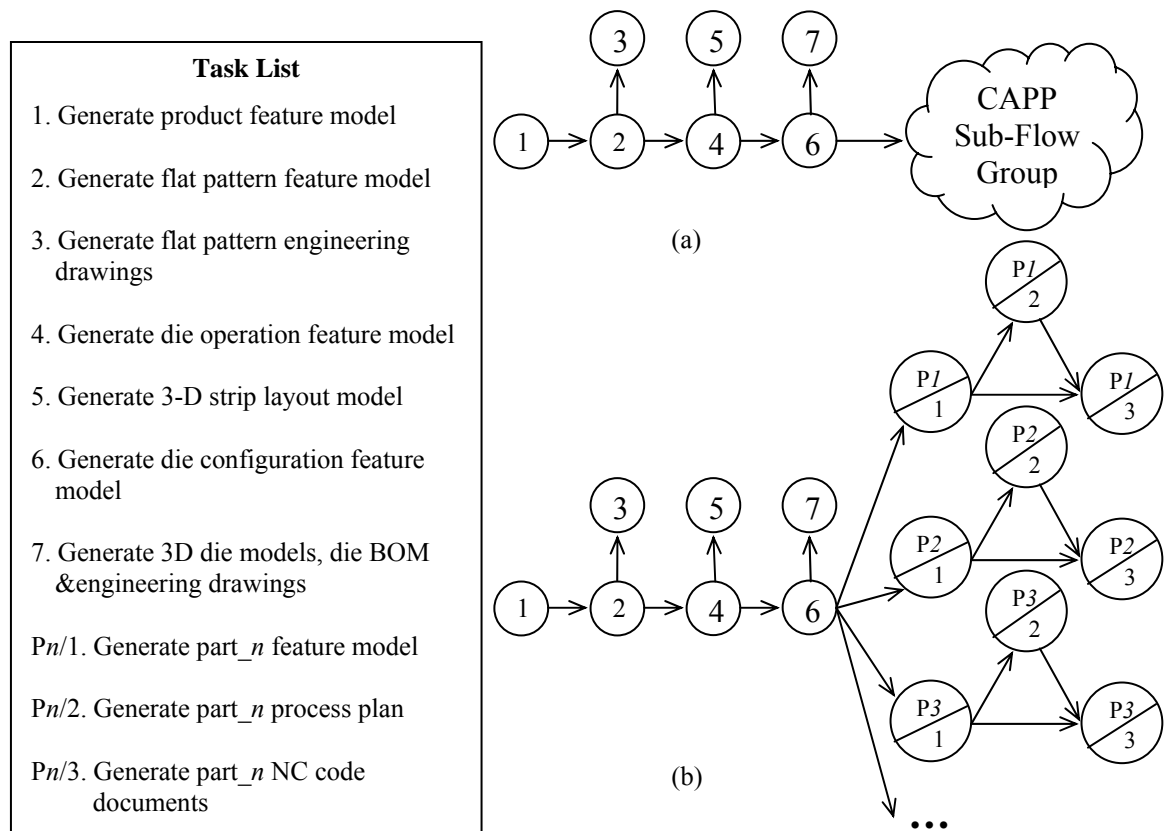


Fig. 6.4. The design flow before (a) and after (b) the CAPP sub-flow is defined

Note that the information structure in Fig. 6.3 is defined in the way that the process management is performed on top of the product data management which is further performed on top of the run-time design transaction management. The “Activity” class which has “Consumed” and “Produced” “DesignObjects” is responsible for bridging

the product data management part and the process management part. The dependence relationships between “*DesignObjectVersions*” in a configuration version are derived from that between “*DesignObjects*” and the later is determined by the activity precedence relationships. Therefore, no such semantics need to be explicitly captured in the information model and further the versioning and configuration management is orthogonal to these dependence relationships owing to the advantages of the database version approach (see Chapter 5).

6.3 Two UML Sequence Diagrams Highlighting the Basic Process Management Functionality

The comprehensive information model shown in previous section defines the meta data schema and provides a static information structure view for the entire system in terms of relevant object types and their relationships. Both the product data management and process management mechanisms are enabled and manifested by this model. To make the defined system more robust, it is usually desirable to examine its internal behaviour and dynamics to trace the sequence of reactions that achieve the specific purposes. One of reasonable ways is to check up all the main operations involved like in the previous chapter where the key is to define individual operations and the interactions between the live objects within the system can be easily recognized from this operation specification. Once the process management functions are involved, however, operations on an object at a certain level are always conducted in the context of a net of complicated interactions with others. It is found that the best way to understand the dynamics in this aspect is to use a formal behaviour modelling technology like the UML sequence diagrams (Fowler & Scott 1997). As such, this section presents two UML diagrams to describe the interactions that occur during two typical process

execution scenarios, with one being simplistic and the other being complex due to the involvement of automatically refining the process configuration dynamically (Fig. 6.5 and Fig. 6.6). From this diagram, it can also make clear what happens in the meta database and design database corresponding to the changes made on the design state in the course of a process execution.

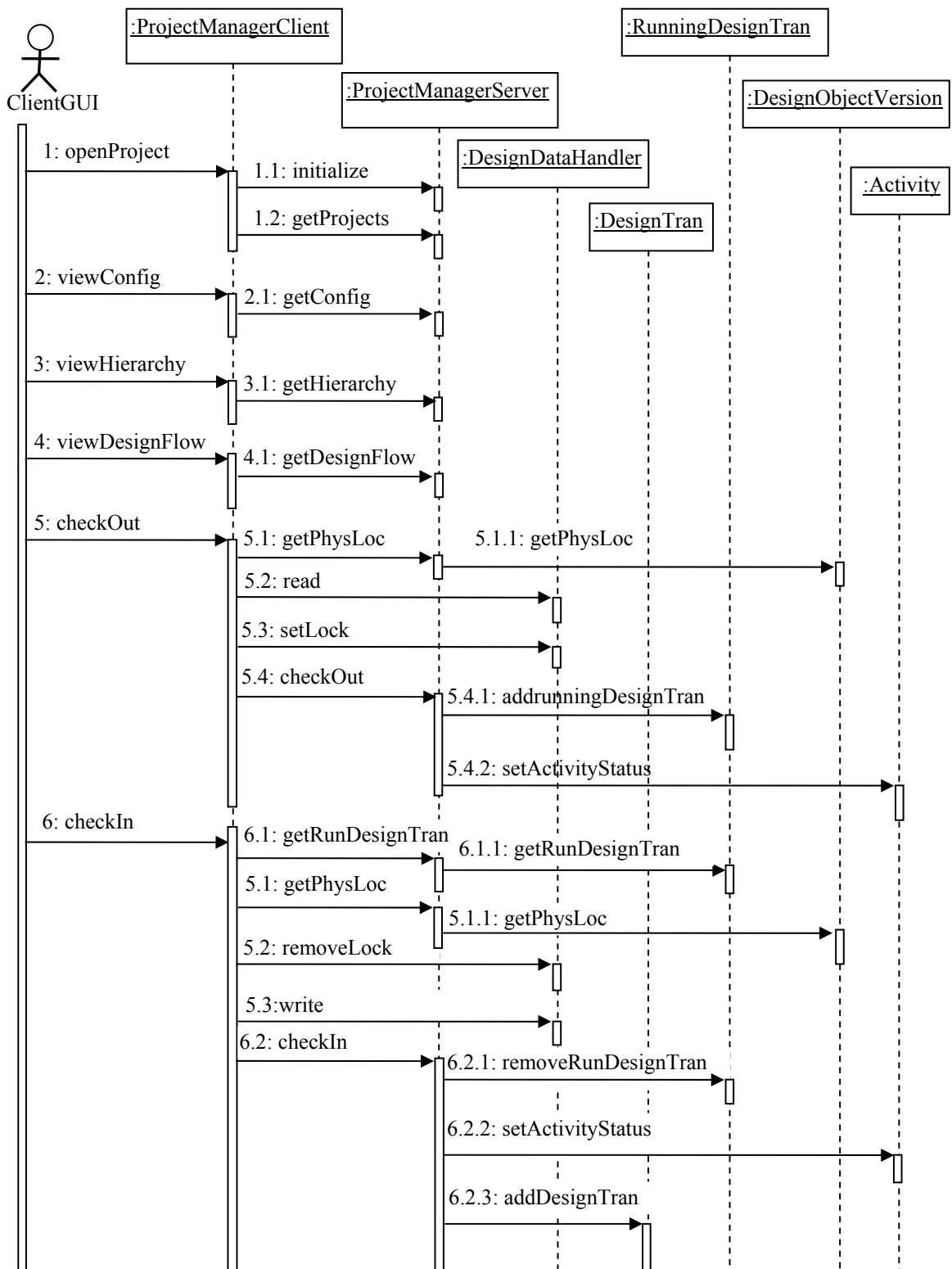


Fig. 6.5. A UML sequence diagram highlighting process management functionality (simple design transaction case)

Fig. 6.5 shows the process execution logic when a simple design transaction is performed. In this scenario, the user opens a project, navigates the project space to a specific product configuration version, views the design objects already generated in the hierarchy and the design state in the design flow browser, finds out the task ready to perform—a half-done task 4 “Generate die operation feature model” (see Fig. 6.4), checks-out the corresponding half-done “die operation feature model” design object, after finishing design, checks-in this design object back to the shared product database. One of important methods in this operation sequence directly relevant to process management support is “*getDesignFlow*” initiated by a “*ProjectManagerClient*” object to retrieve the activity (with corresponding activity status) structure of the design flow belonging to certain configuration version. With this query result, the design flow browser renders an intuitive “coloured” design flow. The user can then easily decide what to do next. Another important method in this aspect is “*setActivityStatus*”, which is invoked to change the activity status in the course to complete the “*checkOut*” and “*checkIn*” operations. All invocations to the methods belonging to objects located on the server side should at first pass a “*ProjectManagerServer*” object which implements a remote interface as a part of the RMI mechanism. The “*checkOut*” and the “*checkIn*” operations trigger a sequence of other corresponding operations following the rules defined in Chapter 4 to maintain the meta data /design data consistency. Since the “*checkOut*” and the “*checkIn*” operations bracket a simple design transaction, a running design transaction is added into the meta database when the design object is checked-out to prevent uncontrolled write. When the corresponding design object is checked-in, a completed design transaction is added into the meta database while the above running design transaction is removed. Note that the design transaction record is monotonously added without remove.

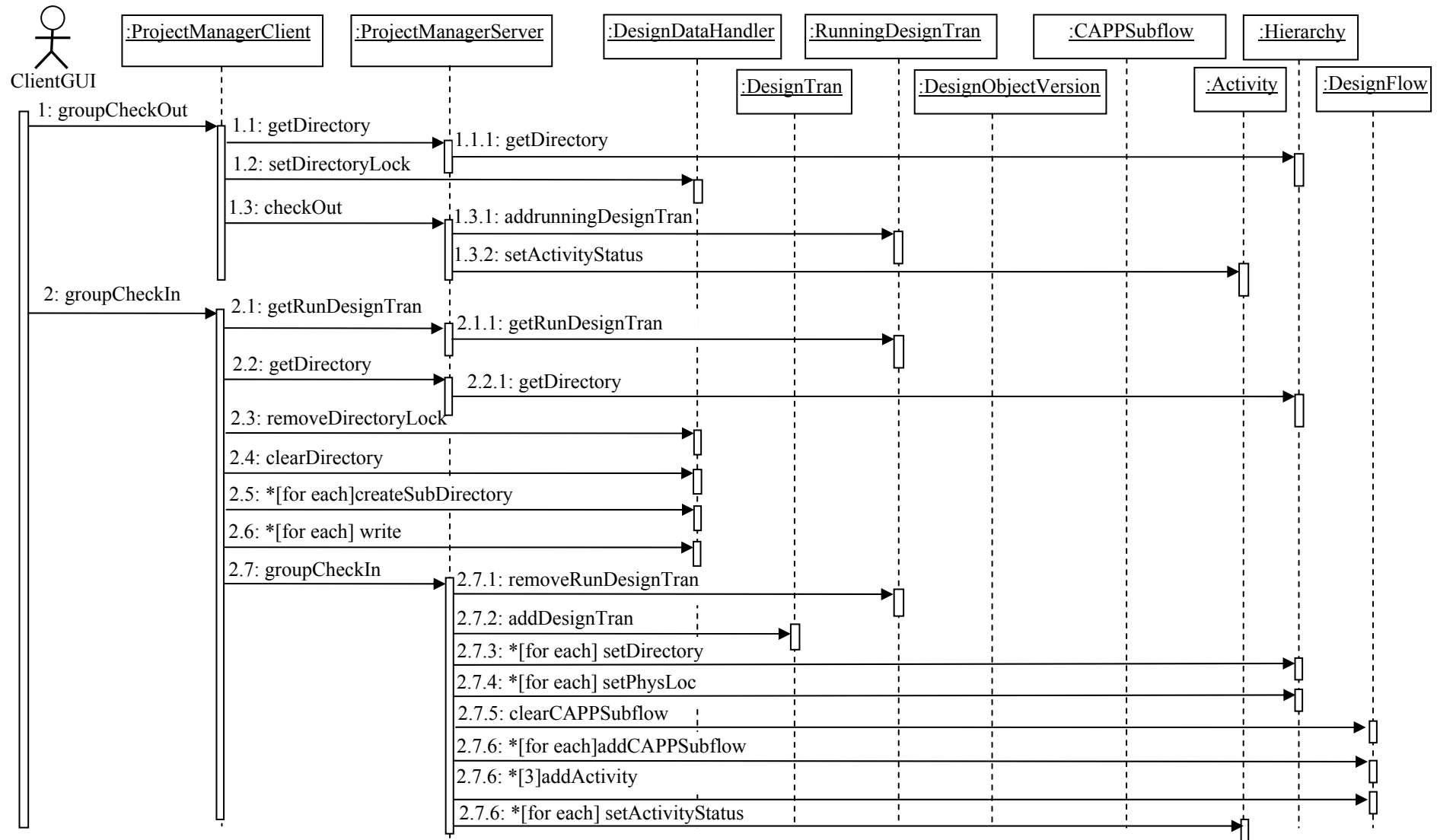


Fig. 6.6. A UML sequence diagram highlighting process management functionality (complex design transaction case)

Fig. 6.6 shows the process execution logic when a complex design transaction is performed. In this scenario, the user has finished tasks 4 and 5 listed in Fig. 6.4 and begins to perform tasks 6 and 7 and all the first tasks for each CAPP group task. All these tasks are performed automatically by an intelligent engineering tool in one turn after the user checks out the die operation model into the working memory. A design transaction bracketed by a physical check-out and a virtual check-in of the consumed design object and a design transaction group bracketed by virtual check-out and physical check-in of the produced design objects are involved in this scenario according to the design transaction model defined in Chapter 3. Since the design transaction on the consumed design object is equivalent to a read-only check-out, operations to perform this design transaction are simplistic and they are not reflected in the diagram. Therefore, only operations related to the design transaction group on the produced design objects are depicted in the diagram. The operations begin from a virtual “*groupCheckOut*” method invocation once the user push a button in the design tool to begin to automatically perform tasks 6 and 7 and others. It is a virtual one because no corresponding design objects are physically checked-out by this operation which only sets locks on them and makes relevant changes on meta data to maintain consistency with the design data. The virtual “*groupCheckIn*” operation begins from retrieval of running design transaction records to guarantee the corresponding running design transaction existent there. It then removes the directory lock and clear former contents in the directory. Sub-directories with constituent design objects in the form of Fig. 5.1 are created in the shared design object repository. Only after the operations on design data are finished, will the operations on meta data begin through invocation of another “*groupCheckIn*” method belonging to the “*ProjectManagerServer*” object. These operations include removal of the running design transaction and creation of

design transaction record, setting of relevant pointers pointing to the physical locations of the sub-directories and design objects and most importantly, reconfiguration of the CAPP sub-flow group with corresponding activity status set. If the CAPP sub-flow group is empty at first, the above operations make a change from what is shown in Fig. 6.4(a) to what in (b). If it is not empty at first, the change is from one type to another type of what is shown in Fig. 6.4(b) with different contents of CAPP sub-flows. The corresponding activity statuses are finally set to complete the “*groupCheckIn*” operation.

It should be noted that the operations involved in the above two scenarios occur within the same product configuration version and thus are all non-versioning according to Chapter 5. It is possible that the check-in operation may result in a versioning transaction to create a new configuration version depending on the user’s option. This aspect of semantics is intentionally neglected due to space limit.

CHAPTER 7

WORKBENCH GUI DESIGN AND SOME EXPERIMENTAL RESULTS

Using the CAX framework approach, a prototype system of a network-integrated feature-driven engineering environment for the progressive die design and manufacturing process has been developed in this study. This chapter gives some look-and-feels about what the system eventually comes into view in front of the end-users. A full-fledged workbench GUI designed to make the internal functionalities approachable is described and some experimental results working on the prototype system through this GUI are reported to further demonstrate the effectiveness of the proposed CAX framework integration approach.

7.1. The Scope of the Demonstration Session

It is impossible, and probably also unnecessary to carry out all implementation details to offer a full-featured physical software system for the purpose of proof-of-concept. Therefore, the prototype system developed in the current study does not intend to reveal all the potential capabilities of the theory presented above. On the other hand, only those distinct from other counter-systems using different integration approaches are selected as the implementation blue print. Especially, emphasis is put on development of the workbench GUI because this GUI offers an interface to access

almost all the significant functionalities attributed to the proposed integration approach. A non-trivial demonstration session running on this GUI was worked out to demonstrate the system capabilities as well as to provide a vehicle to give a feel of the manner how the system works internally. Since the workbench GUI functions as the control panel of the CAX framework which is located at the middle layer between the engineering tools and the global repository, the launch of this application also activates a daemon which intercepts requests from the engineering tools. Therefore, as envisioned in the final industrialized version of this prototype, executions of some most important functions like *check-in* and *check-out* are often originally fired within the engineering tools GUI, not the workbench GUI. However, the workbench GUI also offers a channel to perform these functions provided that the corresponding documents generated by engineering tools are available (this is called flexible, multi-perspective entry to the engineering process (Madni & Madni 1997)). If these functions work well in the workbench GUI, it is easy to make them finally work in the engineering tools GUI through wrappers (see Chapter 4). Hence, it is reasonable for this demonstration session to be all inclusive in the workbench GUI. Similarly, measures were also taken to simplify the running logic underlying the GUI. Some components may be temporarily absent or replaced with alternatives. They are individually experimented in isolation and expected to be encompassed into the full system at the commercialization development stage. For example, the RMI mechanism has been studied with simple examples but not physically encapsulated in the system which enables the demonstration session. Instead, only a local OODB is used to support the demonstration. In this way, the response latency due to network communication when fine-tuning the prototype is avoided and the effectiveness of the result is not affected. This procedure was also applied to the experiment on the jCIFS protocol.

After the above measures were taken, the main concerns of this study were made to avoid being diffused by secondary topics but constantly focused on the novel methodologies. In particular, the demonstration session aims to demonstrate:

- Product configuration and engineering process definition as well as its verification. For any progressive die design and manufacturing projects, the product configuration and process activity structure should comply with some common rules with respect to the composition relationship, activity precedence relationship and design object dependence relationship. Given that this knowledge has been captured, the context-sensitive product configuration hierarchy and design flow controlled by such rules were defined and made virtualized for verification.
- Control of product evolution. In one progressive die design and manufacturing lifecycle, the project is required to be controlled to generate multiple versions at the configuration level and each version evolves from initialization to finalization.
- Virtualization of design state transition (at the document level) along with the product evolution. The evolution of the product is thus perceived intuitively through execution of the workbench application.
- Complete version control and configuration management support. The operations, the specification of which is based on the versioning model presented in Chapter 5, are finally linked to one or a set of mouse or key input actions.
- Control of design change propagation scope. As a part of the version and configuration management functionality, this special issue is highlighted in the demonstration session.

- Dynamic design flow configuration. The CAPP sub-flow is automatically configured in the way presented in Chapter 6, once the framework kernel is informed with relevant information.
- Process execution tracking. Design flow coloring techniques are used to differentiate various states for each constituent activity and thus convey the progress of an executing engineering process to the end-users.
- Application of context-sensitive constraints for executing process. For example, certain operations may not be allowed or may be alerted with important hints before they are performed. Note that not all possible constraints have been explored, and only a few of typical ones are exemplified in the demonstration session.
- Scalability to incorporate general distribution support services. User authentication, off-line work mode support, data replication and other distribution-relevant issues were only superficially demonstrated in the demonstration session.

7.2. Description of the Results for the Principal Demonstration Steps

The developed demonstration session is composed of the following principal steps:

- Application launch, user authentication and work mode selection

The demonstration application was written in Java and developed within the Borland Software Corporation's JBuilder[®] environment. After launching the *run* command, the first window shown to the user allows him to log into the system (Fig. 7.1). If the input personal information is not correct or the network connection is not ready, a re-login request window from which working-on-local-repository (off-line) mode can be selected comes out (Fig. 7.2). When the user enters the main working window either in the on-line or off-line mode, he can change between these two working modes alternatively through pushing one of two buttons in the lower part of the window (Fig.

7.3). Note that the underlying logic to support the user authentication and work-mode selection was not implemented because the former is purely a mundane software coding effort and the latter is relevant to application of the persistent cache technology (Wang *et al.*, 2004) which is out of the scope of this research.

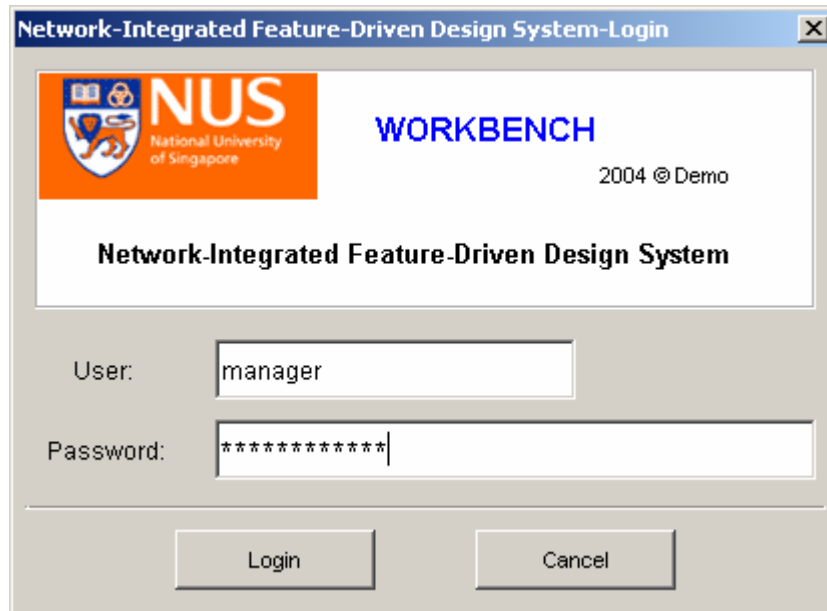


Fig. 7.1. The snapshot of the user authentication window

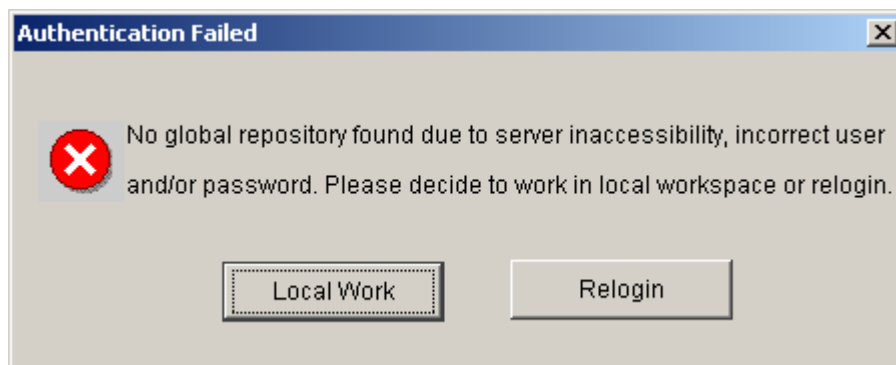


Fig. 7.2. The snapshot of the authentication failing alert window

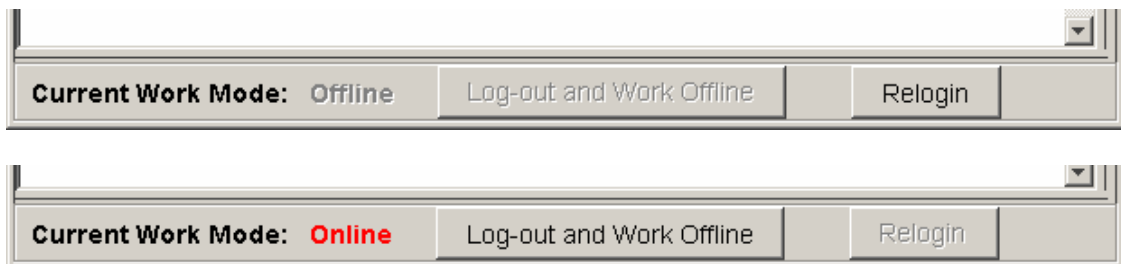


Fig. 7.3. Selection of On-line or Off-line work-mode

- Location of a project to view its constituent configuration versions

Upon entering the main working window, the entries to all the projects managed by the framework kernel is presented to the user for the location of a project to perform related operations on projects. Some control logic is shown below. If no project is selected, all operations apart from the operation *New* which means creating a new project are disabled (Fig. 7.4). Only one project is permitted to be selected and once it is selected, apart from the *Close* operations, all others are enabled (Fig. 7.4). After the project is opened with the *Open* operation, selection of a project is disabled and the project selection record is fixed on the one already selected to alert the user which project is currently opened; the *Close* operation is enabled and the *Open* and *Delete* operations are disabled (Fig. 7.5).

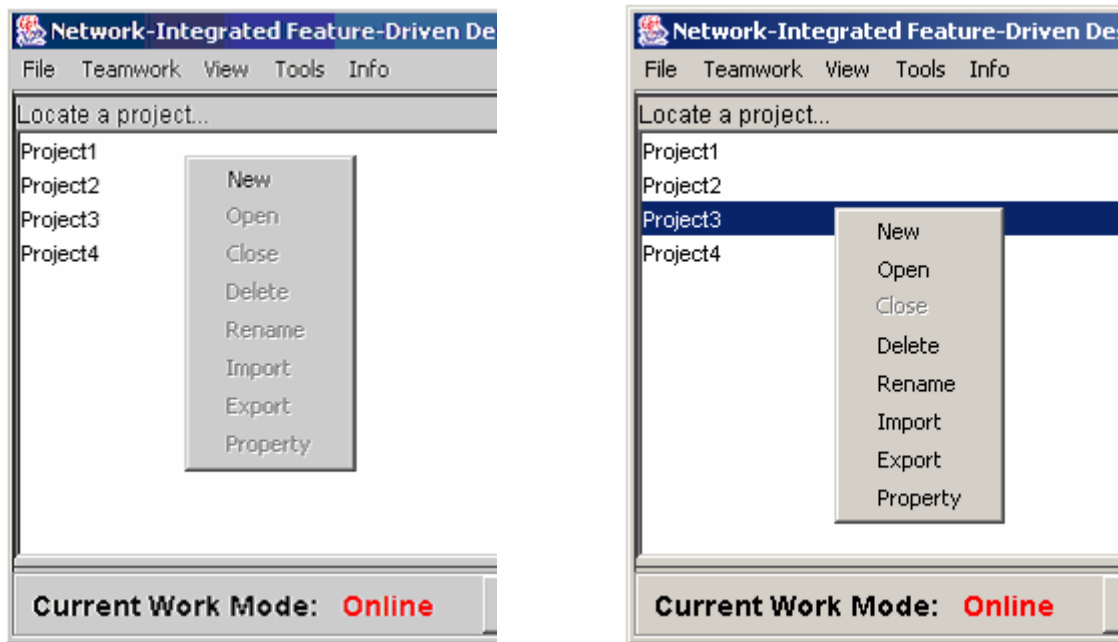


Fig. 7.4. Locate a project

- Location of a configuration version to view its detailed project progress state

Once a project is selected to open, the main working window is broken down to two windows: the project window and the configuration version window. If no configuration version is selected, all operations are disabled (Fig. 7.5). Only one configuration version is permitted to be selected and once it is selected, apart from the

Close View operations, all others are enabled (Fig. 7.5). After the configuration version is opened by the *View Status* operation, in the configuration version window, selection of a configuration version is disabled and the configuration version selection record is fixed on the one already selected to alert the user which configuration is currently opened; the *Close View* operation is enabled and the *View Status* and *Delete* operations are disabled; in the project window, the *Close* operation is also disabled until the *Close View* operation is performed on the opened configuration version (Fig. 7.6). Note that there is no operation on creation of a new configuration version. This is because such an operation is performed implicitly within the operation on new design object version creation which is in turn within the design object *Check-in* operation.

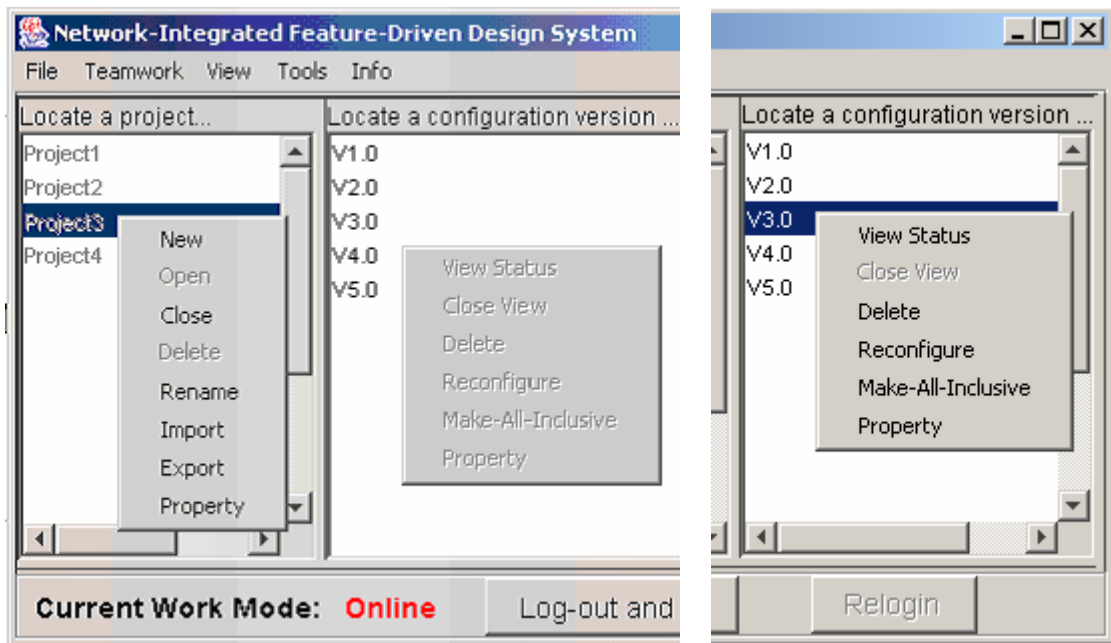


Fig. 7.5 Locate a configuration version

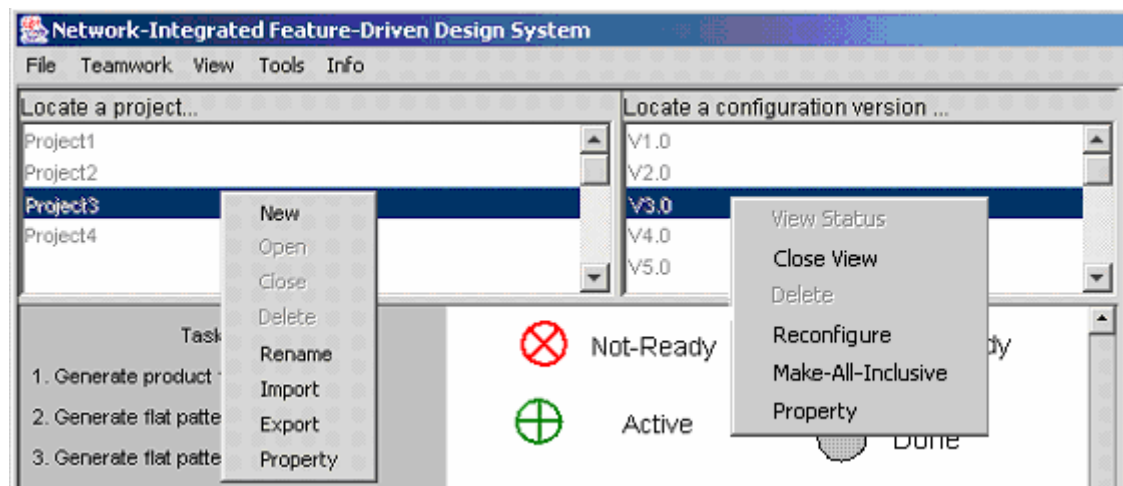


Fig. 7.6 Open a configuration version to view its running design flow and composition hierarchy

- Launch of the design flow window and the composition hierarchy window

Once a configuration version is opened by a *View Status* operation, the working window is enriched with another two most important windows: the design flow window in lower-left corner and composition hierarchy window in the right side (Fig. 7.7). The design flow window is designed for visualizing the design flow status. It consists two parts: the annotation and viewing control panel and the view port to render the 2D design flow drawing. Specific tasks (activities) are represented by numbers when rendering so as to keep the drawing neat. The drawing consists of nodes representing a design activity and directional lines representing the precedence relationship. The nodes is “coloured” into any one of four patterns representing four types of activity status. In order to obtain the best virtualization effect, the drawing can be panned and zoomed and the node can be moved and resized. Further, the control panel can be hidden to allow more space for drawing rendering. The composition hierarchy window is designed for visualizing the project composition and progress status in terms of design objects (versions). It is also used to receive users’ operations on design objects, especially the check-in and check-out operations to drive

the engineering process to evolve. The structural design objects correspond to no physical documents. They are used as directory to reflect the composition relationships. The physical design objects are also “coloured” to differentiate between causal, unchanged, pre-created resultant and updated resultant design object versions. The operations are activated through a pop-up menu attached to a selected design object. The *Check-in* menu item has two branches to make it further refined whether it will create a new version or simply overwrite the old value. For the design objects with the *IsProactive* attribute (see Chapter 5) being false, such as the *Die Configuration Feature Model*, the *Create New Version* operation is always disabled.

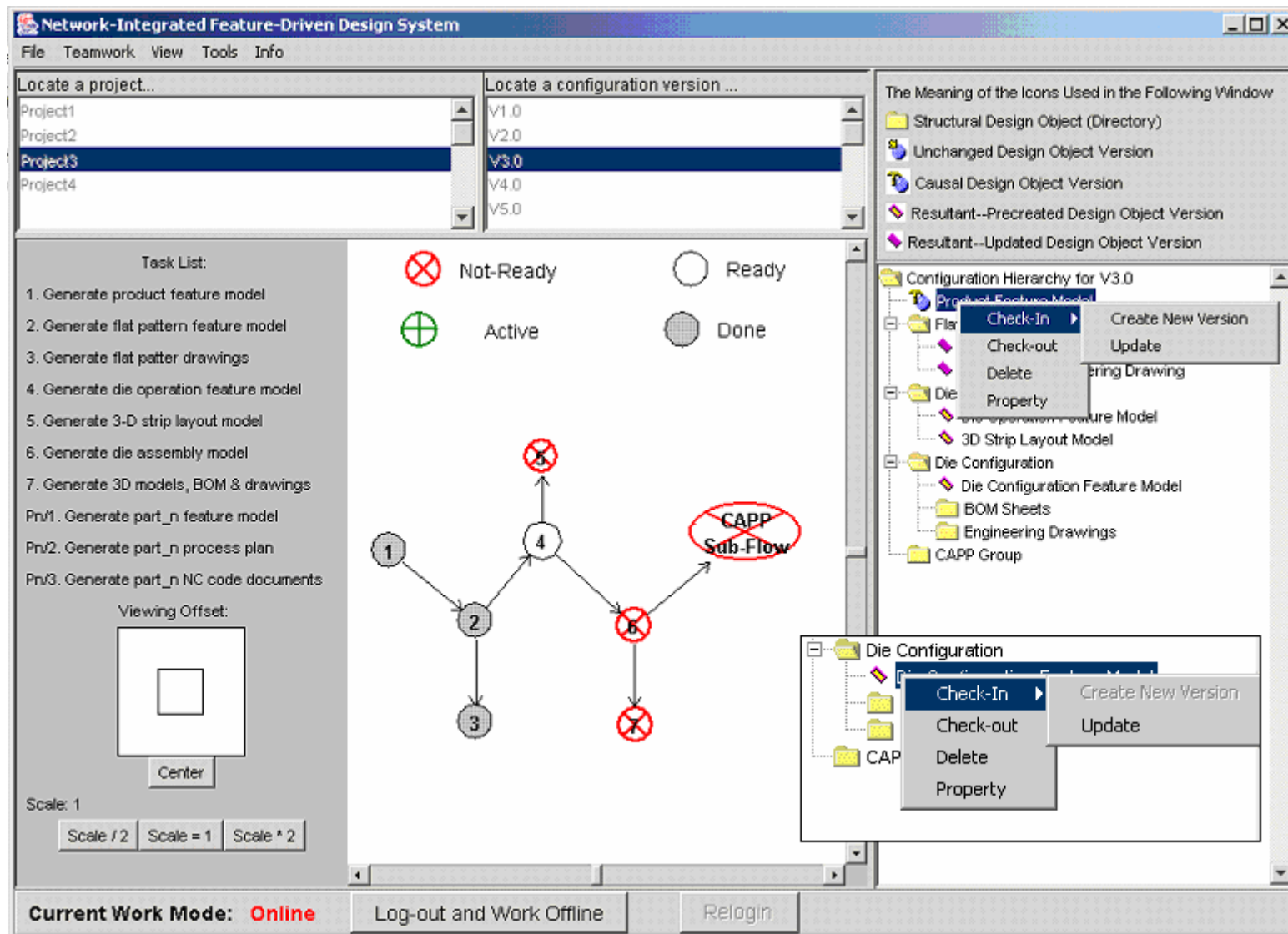


















Fig. 7.7. Composition of the overall working window

- Simulation of the evolvement of a configuration version

The configuration version under simulation is *V3.0* belonging to *Project3*. Its causal design object is *Product Feature Model*. All other design objects are resultant design objects which are pre-created with *nil* value at first and then updated via non-versioning transaction. The simulation begins from an intermediate state at which the Flat Pattern Feature Model and Flat Pattern Engineering Drawing have been finished and the activity 4 “Create die operation feature model” is ready for performing. The simulation result is summarized in Table 7.1.

Table 7.1 Evolvement of a configuration version

Sequence No.	Operation on	Name of the operation	Changes occurring in the Design Flow	Changes Occurring in the Hierarchy
1	Die Operation Feature Model	Check-Out	 → 	--
2	Die Operation Feature Model	Check-In ... Update	 → 	 →  for Die Operation Feature Model
3	3D Strip Layout Model	Check-In ... Update	 →   →   →   → 	 →  for 3D Strip Layout Model

- Grouped check-in and configuring design flow dynamically

With the evolvement of the selected configuration version, when it comes to the point to check-in the Die Configuration Feature Model, this means that the die design tool begins automatically configuring the progressive die, and a range of documents are generated. According to the augmented design transaction model presented in Chapter 3, these documents should be checked-in in a group. So when a check-in operation is performed upon the Die Configuration Feature Model, a dialogue window springs out to alert the user to make sure it is ready for performing such an operation (Fig. 7.8). After it is confirmed, the condition for refinement of the CAPP sub-flow is satisfied, it then extends to the activity level. Correspondingly, the hierarchy is also made to

incorporate all the CAPP documents with the Part Feature Models being generated by the current operation and others being pre-created with *nil* value (Fig. 7.9).

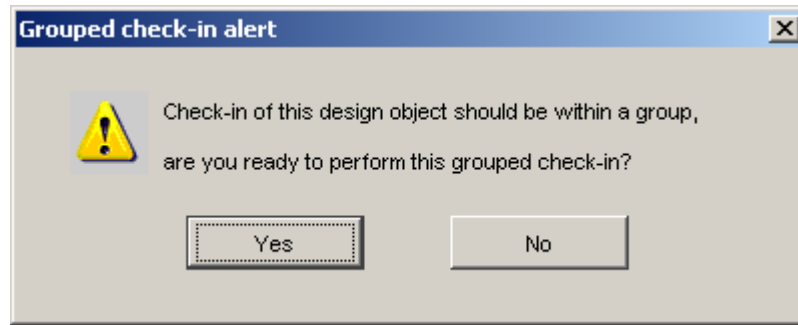


Fig. 7.8. The grouped check-in alert dialog

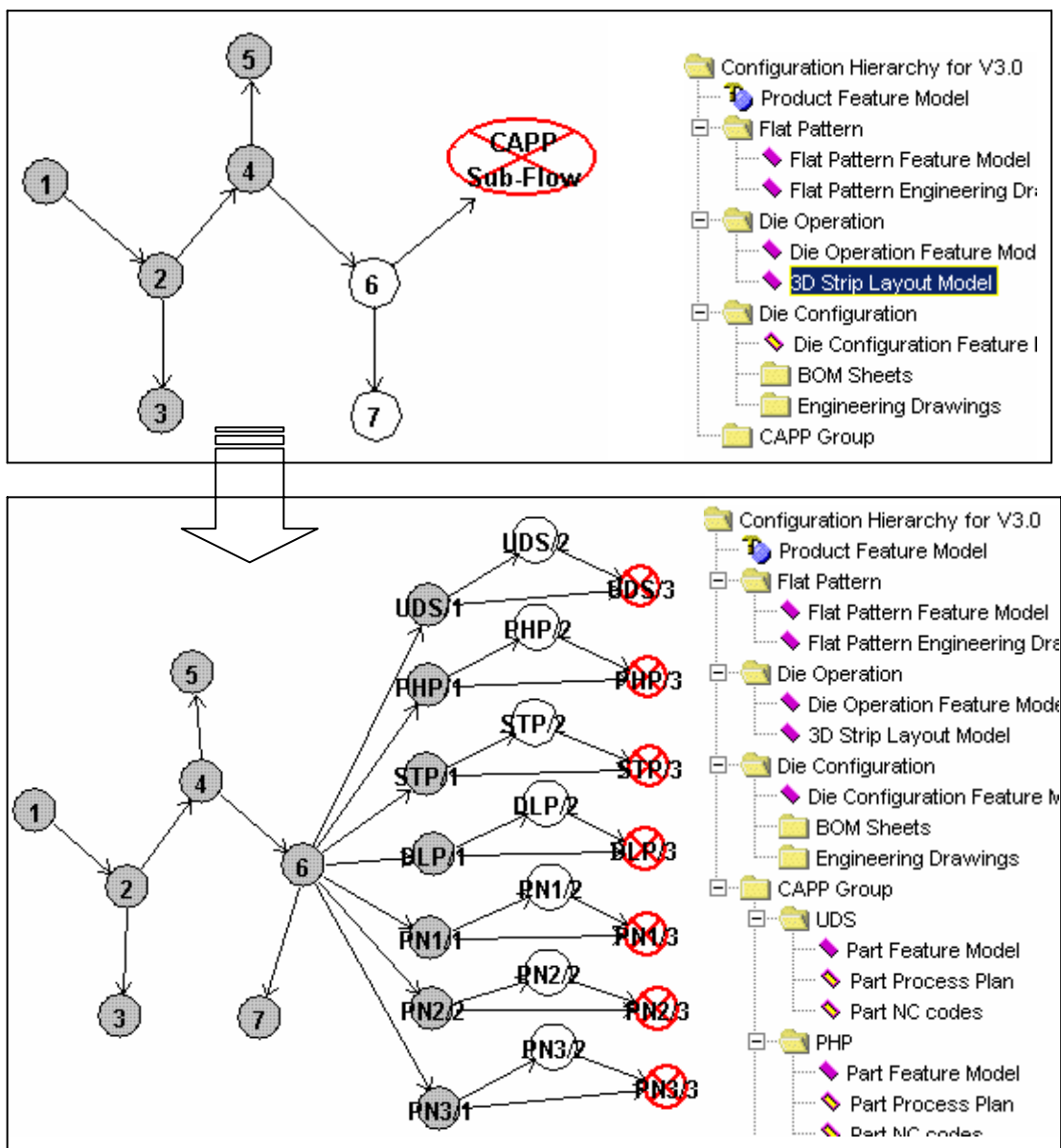


Fig. 7.9. Design state change caused by a grouped check-in

- Concurrently performing multiple CAPP engineering activities

When the configuration version evolves to the stage of performing CAPP activities, many components are involved, but the design tasks for each component are identical, and further, the activity group for one component is independent of that for other components. Fig. 7.10 shows the design state after a couple of check-out/check-in operations from the above design state. Two engineering activities are concurrently performed at this design state.

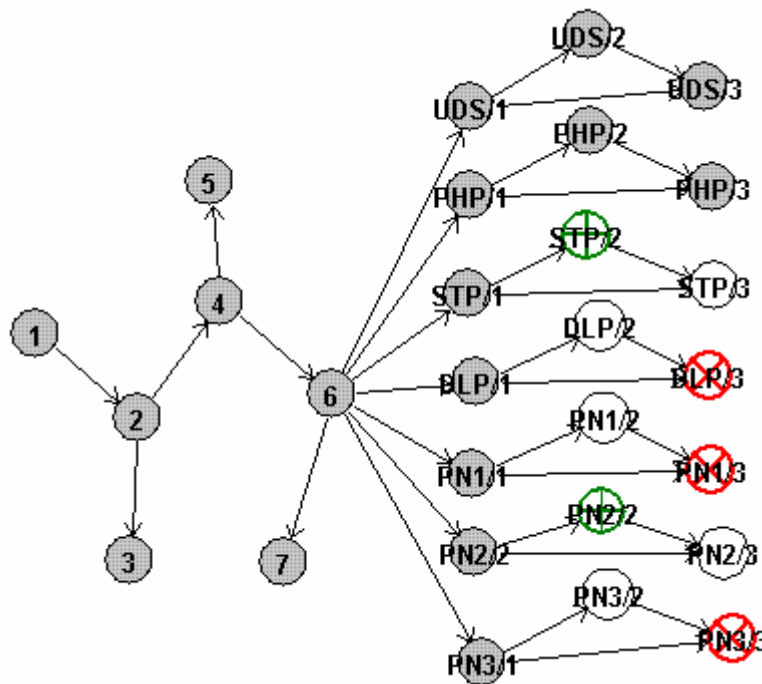


Fig. 7.10. A design state at which two activities are concurrently performed

- Creation of a new configuration version

Suppose right at the above design state, an engineer (may be different from those working on CAPP tasks) comes up with a new idea to try an alternative for the strip layout design and makes a decision to generate a new version of Die Operation Feature Model. This needs to firstly check-out the Die Operation Feature Model, make certain modifications, and then check it back into the global repository to create a new version. When performing check-out, no changes occur apart from that on activity 4, the state

of which is transitioned from “done” to “active”. When performing check-in, before a new version for this design object is created, a new configuration version is created through a window shown in Fig. 7.11. After the user input required information for the coming new configuration version, such as its version ID, its unchanged and resultant design object versions, a new configuration version is created, which can be observed in the configuration version window where a new configuration version record identified by the ID input a moment ago is added. The design state in terms of running design flow and composition hierarchy for this newly created configuration version is shown in Fig. 7.12.

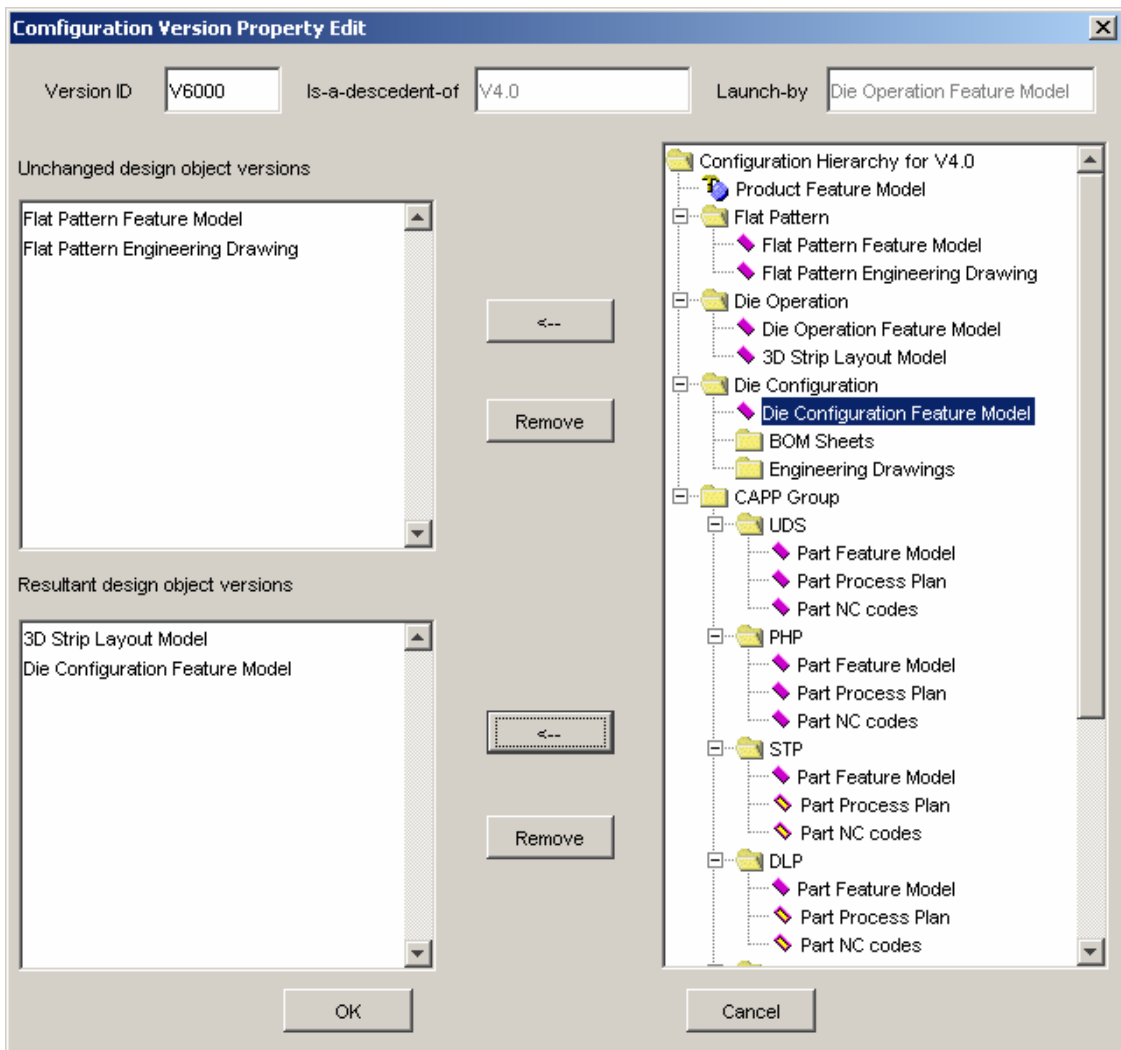


Fig. 7.11 Creation of a new configuration version

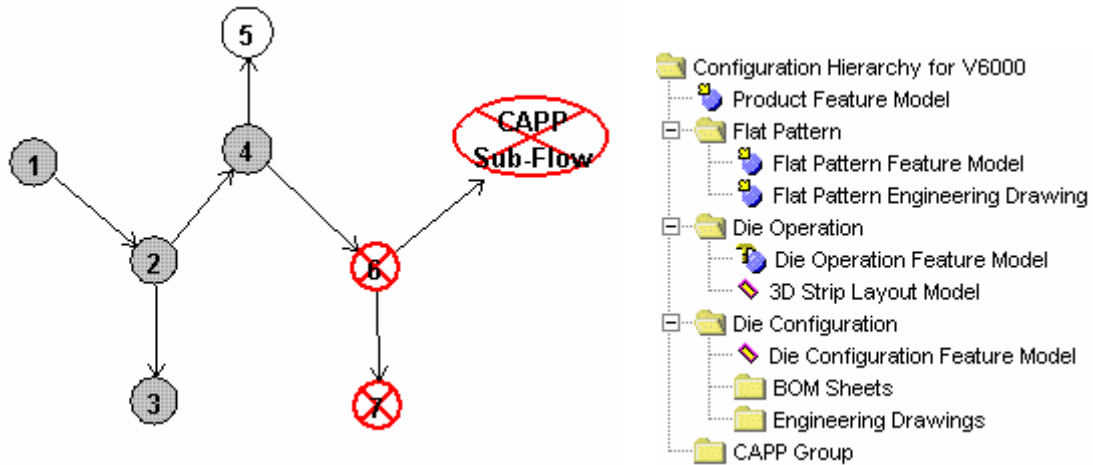


Fig. 7.12. The newly created configuration version

7.3. Discussions

7.3.1. Evaluation

The research prototype developed in this study was primarily employed for demonstration sessions. It is not a complete system implementation but focuses on what can be expected in the user interface and what novel capabilities can be reaped from the corresponding fully-specified system. Moreover, it is intended to be replaced by a stable and comprehensive system in the future. However, this does not mean that these capabilities are pending. On the other hand, they should be considered as determinative with adequate confidence since the system specification in the aspect of data structure and operation sequence has been evaluated by the proven UML models. Eventually realizing a comprehensive system is mainly a matter of time because the system specifications and the UML models have been developed in this study. The current prototype has about 4,000 lines of codes excluding adaptive reuse of about 40,000 lines of open-source codes for virtualization of the design flow. It is estimated that the number of the lines of codes for the complete system may be of several times of the current one.

The GUI design itself is also significant since the design state information maintained in the framework kernel will become useless if it is not adequately virtualized in the user interface. Process management assistance firstly means intuitively informing the user about the design state so that he can avoid loss of track in the process and can immediately perform what is exactly required to be done at the moment with all required resources available. The demonstration session described above may be able to strengthen this statement.

Even if the complete system is achieved finally, it is still difficult to obtain reasonably justified statements such as “introduction of the management system x has improved our productivity by y percent” (Westfechtel 2000). Therefore, evaluation based on the examination of the individual system capabilities instead of waiting for comprehensive experiment results in the field is probably the only option that can be adopted to demonstrate the effectiveness of methodology. The following paragraph summarizes the system capabilities in supporting concurrent engineering and presents some additional predictable capabilities which is not demonstrated in the demonstration session but can be obtained with relatively uncomplicated efforts.

7.3.2. Concurrent Engineering Support

Section 7.2 has depicted the way the system supports performing multiple CAPP engineering activities concurrently. This only reflects one aspect relevant to such capabilities as assisting the execution of the CE strategy. Another two aspects include:

- Facilitating information sharing and exchanging. All the product development results-related information including that for the intermediate premature versions is stored in the global repository and can be easily retrieved. This allows the maximum extent of concurrency among different development activities in a predictive way based on the information available and the down-stream engineering activities may be launched in advance without unnecessarily waiting for the release of the decisive design.
- Facilitating concurrent performance of engineering activities belonging to different configuration versions. Because the versioning scheme is set at the configuration level, this logically makes the works on one configuration version to be relatively independent of that on another. Therefore, several engineers may be allotted to work on a number of different versions simultaneously in a controlled way. The achievement of an optimum solution based on comparing multiple alternatives may come earlier than the current practice which has no similar versioning control and configuration management support.

7.3.3. Further Predictable Capabilities

Based on the ideas gained from the study performed by researchers such as Madni & Madni (1997), the current system has the potential to be equipped with capabilities such as:

- Creation and update of project progress reports

In the engineering practice, an engineer may often be required to produce a progress report urgently by some persons at a higher managerial level. He then consults relevant

design members and collects relevant information dispersed in different places. After the report comes to the person who is calling for it, he may just say, “OK”, and then glances over the report and may shelf the report casually, without knowing where it has been kept. Yet the designer may be required to repeat the same work patiently again and again. The current system has good potential in relieving the engineers from such mal-practices. Since almost all the relevant information has been maintained in the global repository, creation of progress reports may be realized by just a push of a button and some small additional effort on refining the draft automatically generated by the system.

- Replay of design and process history

For running an engineering process, even for a sign-off project, it is possible to replay the whole design process history to show how the current state is reached from initiation. This is probably especially important for training novice engineers.

- Recovery from engineering process “breakdowns”

For some reasons, for example, if a key engineer in a project leaves the company, an engineering process may be interrupted suddenly. The current system can help the recovery of the halted process easily by re-allotting the role to a replacement engineer, provided that the management role is added. The newly appointed engineer can be easily updated with the knowledge about the history and the current state of the process as if he has been participating in the project from the beginning.

- Context-sensitive designer guidance

The above paragraphs have shown context-sensitive constraints which prevent the users from performing error-prone operations. Similarly, context-sensitive designer guidance can be added to the current system. This handy guidance should outperform the use of a thick design manual.

CHAPTER 8

CONCLUSIONS

The extensive capability of a widely-known system integration approach centered on a CAD framework for EDA has driven this study to use an analogous approach centered on a CAX framework for developing network-integrated engineering environments in the area of manufacturing engineering. This chapter concludes the study that has been presented and discussed in this thesis.

8.1. Research Contributions and Discussions

To sum up, the main contributions of this study include:

- Comprehensively characterizing the feature-driven engineering process, a promising area to apply the CAX framework approach. This has been regarded as the starting point to develop a network-integrated engineering environment, which explicitly takes into account the characteristics identified. These characteristics themselves are significant in many other aspects. For example, the identified characteristics reflecting the model-model relationships between two interdependent step-processes in terms of equations (3.5) and (3.6) can be expansively exploited. They can improve the understanding why design automation is possible, what is the limit of design automation and how to design mechanisms to implement design automation or design change propagation automation.

- Development of an integration architecture based on the CAX framework approach. By adaptively using the concepts and principles found in the CAD framework approach, the architecture is incrementally built up beginning from identification of the functional requirements of the CAX framework. Two types of integration functions, the product data management and the process management, are provided by the framework. This makes the framework comparable to a light-weight PDM/WM module for the participating CAX tools. OO strategy is used to develop the framework and a two-step implementation roadmap is recommended. Firstly, a “skeletal” framework is derived while a range of basic implementation decisions are made. The second step is to develop the product data and the process management model as the management database schema, based on which the information structure of the whole framework is developed.
- Development of a version control and configuration management model supporting the management of design change propagations. A very broad spectrum of semantic and operational issues is addressed.
- Development of a process control model which views a feature-driven engineering process as a semi-structured design flow allowing dynamic specification while process is executing.
- Development of a prototype which uses the above architecture and product data and process management models. The prototype is a network-integrated engineering environment for CAD/CAM of progressive dies. It has been the vehicle for validating many of the relevant conceptions and proposals.

Based on the experience to develop the prototype system and the completion of a demonstration to illustrate its capabilities, it can be concluded that: the CAX

framework integration approach can turn a collection of distributed but logically related CAX tools into effective user-friendly environments with value-added integration functions, such as product data management and process management; It is therefore recommended for CAD/CAM system developers to adaptively use this approach if their targeted design-to-manufacturing process can be roughly classified as a feature-driven engineering process.

In general, the key points to the main procedures for applying the CAX framework approach can be briefly summarized as following:

- Integration should begin from adequate process decomposition, analysis, modeling and re-engineering. IDEF0 activity modeling is the most important tool to carry out this mission.
- From the global view, the network-integrated engineering environment developed based on the CAX framework approach is composed of a set of CAX tools and the CAX framework, which further consists of a workbench application accessible by all the tool users, the framework kernel, a management database and the raw design data repository.
- Development of the framework can take two steps. The first step is to make all implementation decisions to conceptualize a “skeletal” framework with the management database schema being empty. Such decisions include those dealing with how to interface design tools and the framework (simply how to wrap), what roles are allotted to the framework, how to partition the framework functions between the client side and the server side, what languages are used to program the framework kernel, *etc.* The second step is to develop the management database schema or relevant information models and further make the database coherently co-

work with other components in the framework. For achieving the coherence, the information models for database schema (state-permanent part in the information models) and those for describing the working modules in the system (state-transient part in the information models) should be linked together for performing system analysis and making the adequate decisions. Object-orientation should permeate the full system development process from beginning to end. For example, OO programming languages, OO modeling methods, ODBMS, distributed object technologies are recommended to be used wherever relevant.

- The information models for database schema typically include two parts: one for realizing PDM, the other for process management. The full information modeling course should be incremental. A good modeling sequence works like this: PDM at first, process management and then, overall at last.
- Examination on how the CE strategy is supported is another factor in need for consideration throughout the whole system development process even including the system evaluation and improvement phase.

It is important to note that these points are very compatible with those comparative points made in Chapter 2 based on a comprehensive survey on principal aspects driving system integration from design to manufacturing

Compared with the CAD framework approach which became mature in the 1990's, the CAX framework approach makes full use of the latest system analysis strategies, such as OO, and relevant information technologies, such as the distributed object technology RMI.. Especially, building the CAD framework includes three steps to incrementally build the information architecture, the component architecture and the

implementation architecture (Wolf 1994) because various system definition and implementation primitives have to be used. On the other hand, building the CAX framework is recommended to take two easy-to-follow steps because a common primitive, or object, can be used. Of course, it is possible for the CAD framework to evolve to also use the OO methods. This makes both the CAD framework and the CAX framework have no radical methodological differences apart from in that the CAD framework is applicable to EDA, while the CAX framework to manufacturing. To the author's best knowledge, there is no literature that deals comprehensively with CAD framework in the OO context. Therefore, the current effort to develop an OO CAX framework for manufacturing may be useful to develop an OO CAD framework for EDA. In brief, while comprehensively making use of the OO technologies, the current CAX framework emulates the CAD framework which already has a strong theoretical foundation. This allows the system developers to easily specify the desired integration infrastructure for a range of dispersed, but logically related, CAX tools.

The effectiveness of the CAX framework approach is probably due to its intelligent ability to address the integration problem like a human manager who is responsible for maintaining global process cohesion between individual sub-processes carried out with the help of a set of isolated engineering tools. Specifically, in order to integrate an engineering process within an enterprise, or even a virtual enterprise, it is natural to (logically) centralize all the distributed engineering data, manage them at a high abstract level and help users to drive the engineering process through a process knowledge-enabled utility based on the design status captured by a management database. The low level data consistency problem is left to be locally handled by the individual engineering tools. In this way, the requirement of remote computing

resources for the goal of information sharing and exchange can be minimized and thus the system performance can be optimized. The effectiveness of the CAX framework approach is also reflected in its conformance to the CE philosophy, which has been elaborated in Chapter 7.

8.2. Limitations

There are some limitations that can be observed in this study. The details are depicted as follows.

- One important limitation is that the demonstration session presented in Chapter 7 is brief and the demonstration steps involved in this demonstration session are loosely related. One solution to this limitation is to broaden the scope of the demonstration. Two demonstration sessions would be adopted accordingly. The current demonstration session including eight demonstration steps could be classified into “an introductory demonstration session to show the basic system functions”. Another demonstration session, “a sequential demonstration session corresponding to a progressive die design process scenario”, would be added. To ensure the sufficient complexity, this scenario can be defined as a part of the versioning scenario in Chapter 5, specifically, versioning step 1: propagation of a design change to generate Con_2 from Con_1 . With the help of this new demonstration session, the thesis can be expected to illustrate more explicitly the key concepts presented and implemented.
- One of additional limitations is that several conceptually feasible advanced functions were not implemented and tested in the currently developed CAX framework, thus

the potential of the proposed approach was not fully demonstrated. These advanced functions, such as cooperative engineering transactions, project-level activities management, reusable CSCW-like services, *etc.*, were mentioned where appropriate, but not thoroughly studied. It also needs to be pointed out that further technological developments are required to offer a full-featured system for industrial application based on the prototype.

- Another limitation is that the current CAX framework was not strictly designed as a configurable, reusable, ‘semi-complete’ application that can be specialized to produce the prototype-like custom applications. Instead, it was directly modeled as based on the application instance schema, rather than a meta-schema, which is the schema of the schema and can be used to generate the above instance schema by the system interactively and semi-automatically at the framework configuration time. This decision is attributed to the fact that more application contexts beyond the progressive die design and manufacturing process should be investigated before a general meta-schema can be developed. If based on only one specific application context, the developed meta-schema may not have generic representative ability. It is believed this limitation does not affect the generic sense of the approach: if it is used to develop another engineering environment applicable to a new context, the system information modeling schema can be easily achieved by adapting those given in this thesis.
- Yet another limitation is that the current CAX framework only supports the traditional PDM-like integration at the coarse-grain level, *i.e.*, the file (design object) level. The tighter integration at the lower information level, such as the feature level,

is not explored. The coarse-grain integration strategy is compatible to the current process decomposition strategy to divide the overall information space (apart from the final engineering outputs) or feature space into a collection of sub-spaces. Each sub-space corresponds to an isolated feature-based model, such as the product feature model, the flat pattern feature model, the die operation feature model, *etc.* The study does not demonstrate whether it is possible to unite all the isolated feature-based models into one unified feature-based model based on a re-designed feature taxonomy covering the overall feature space. If the unified feature model exists, the CAX framework can be built on this model. Further, carrying out the full die design process means incrementally achieving this single feature-based model rather than a set of independent feature-based models. In this way, all the design activities are by nature tightly integrated at the feature level.

8.3. Future Directions

There are several directions for future research with respect to development of the network-integrated feature-driven engineering environment based on the CAX framework approach, apart from those immediate system improvements described in Chapter 7.

- One important research issue is to refine the functional requirements of the network-integrated system for the intended application area – the progressive die design and manufacturing, from new perspectives, such as asynchronous collaboration. Since accomplishing a progressive die design and manufacturing project entails extensive asynchronous collaborations among a multi-disciplinary work team, it is strongly desirable for the system to make more efficient the asynchronous communication of

design changes among the team. The current CAX framework attempts to mainly provide product data and process management assistance for the end-users individually. Storing and managing the design changes among the team members was not explicitly taken into account. If the CAX framework is made to explicitly support asynchronous collaboration, the down-stream engineers can collaborate with their up-stream partners more efficiently. Consequently, the collaboration characteristics of the progressive die design and manufacturing processes need to be investigated. Special design transaction models, such as the cooperative engineering transaction model, may be required to be developed and adopted. The most appropriate selection of implementation technologies needs to be made based on a comparison among a pool of alternatives, including those adopted in this study.

- Another future research issue is to re-construct the information architecture of the CAX framework using a medium-grain information primitive, i.e., feature. The overall die development process needs to be re-engineered and the process decomposition needs to be refined to allow generation of one unified feature model instead of a set of smaller feature models for one project. The design change propagation mechanism currently based on the concept of configuration version can be easily made available within the unified feature model because a configuration version corresponds to a version of the unified feature model. Further, the dependence relationships among feature-based models can be precisely represented by the constraints among the features in the unified feature model. Therefore, automatic design change propagation is possible if the constraints is captured and implemented in the unified feature model. The UOF (Unit Of Function) concept (Urban *et al.*, 2000) may be useful in this case to view both the medium-grain

features and the coarse-grain engineering outputs as UOFs. Correspondingly, the object-relational (instead of OO) database management system like the Oracle® database system is recommended to be adopted because of its strong ability to uniformly represent the information elements at different levels. The management database can also adopt Oracle® database system. Therefore one common database system, instead of two different ones, can be used.

- Yet another future research issue is to re-construct the implementation architecture of the CAX framework using web service technology (Tamine & Dillmann 2003, Molinari *et al.*, 2004, Wu, *et al.*, 2004) which has been rapidly developing since a few years ago. More development efforts are required to be spent on the server side. It can become more explicit to observe that the system built in this way is working on the network and the remote resources/services are exploited by local applications or users when needed.
- Apart from further development based on the current study on the CAX framework for progressive die design and manufacturing, it is also valuable to investigate the applicability of this approach in other areas. With more application contexts studied, one can then consider making the CAX framework configurable so that its information architecture can be generated according to different application contexts from a high level meta-schema instead of being predefined in advance at the instance schema level.

REFERENCES

1. Agrawal, R. & Jagadish, H.V. 1989. On correctly configuring versioned objects. Proceedings of the 15th International Conference on Very Large Databases, Amsterdam, The Netherlands (VLDB '89), August 1989: 367-374.
2. Algeo, A.M.E., Feng, C.S. & Ray, R.S. 1994. *A State-of-the-art Survey on Product Design and Process Planning Integration Mechanisms*. An Internal Report of National Institute of Standards and Technology: NISTIR 5548.
3. Ahmed, R. & Navathe, S.B. 1991. Version management of composite objects in CAD databases. Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data, Denver, Colorado, USA: 218-227.
4. Anonymous 1. Web publication available at: <http://jcifs.samba.org/>
5. Anonymous 1998. *Product Data Management: the Definition, an Introduction to Concepts, Benefits and Terminology*. CIM-data.
6. Arnold, F. & Podehl, G. 1998. Best of Both Worlds - A Mapping from EXPRESS-G to UML. *Lecture Notes In Computer Science*. Selected papers from the First International Workshop on The Unified Modeling Language «UML»'98: Beyond the Notation. 1618: 49-63.
7. Baldwin, R.A. & Chung, M.J. 1995. Managing engineering data for complex products. *Research in Engineering Design - Theory, Applications, and Concurrent Engineering* 7(4): 215-231.
8. Beech, D. & Mahbod, B. 1988. Generalized version control in an object-oriented database. Proceedings of the 4th IEEE International Conference on Data Engineering, Los Angeles, CA, USA, February 1988: 14-22.

-
9. Black, R. 1996. *Design and Manufacture: an Integrated Approach*. Macmillan Press Ltd. Basingstoke, Hampshire.
 10. Borja, V., Bell, R. & Harding, J.A. 2001. Assisting design for manufacture using the data model driven approach. *Proceedings of the Institution of Mechanical Engineers, Part B (Journal of Engineering Manufacture, 215(B12): 757-1771*.
 11. Bounab, M. & Godart, C. 1997. Tool integration in distributed environment: an experience report in a manufacturing framework. *Journal of System Integration 8: 31-45*.
 12. Bounab, M. & Godart, C. 1998. Tool integration in distributed environments: an experience report in a manufacturing framework. *Journal of System Integration 8: 31-51*.
 13. Bronsvort, W.F. & Jansen, F.W., 1993. Feature modelling and conversion - key concepts to concurrent engineering. *Computers in Industry 21(1): 61-86*.
 14. Carnduff, T.W. & Goonetillake, J.S. 2004. Configuration management in evolutionary engineering design using versioning and integrity constraints. *Advances in Engineering Software 35(3-4): 161-177*.
 15. CFI Architecture Technique Subcommittee. 1990a. *Suggested Framework Problem Statement*. CAD Framework Initiative.
 16. CFI Architecture Technology Subcommittee. 1990b, August. *CAD framework users, goals and objectives, Version 0.91*. CAD Framework Initiative.
 17. Chen, B.T.F. 1982, September. ROMAPT: a new link between CAD and CAM. *Computer Aided Design 14(5):261-266*.

18. Chen, Y. & Hsiao, Y.T. 1997. A collaborative data management framework for concurrent product and process development. *International Journal of Computer Integrated Manufacturing* 10(6): 364–376.
19. 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.
20. Cheok, B.T. 1998. *Intelligent Techniques for Progressive Die Design*. PhD thesis, National University of Singapore.
21. Cheok, B.T. & Nee, A.Y.C. 1998 (a). Configuration of progressive dies. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*. AIEDAM, 12(5): 405-418.
22. Cheok, B.T. & Nee, A.Y.C. 1998 (b). Trends and developments in the automation of design and manufacture of tools for metal stampings. *Journal of Materials Processing Technology* 75(1-3): 240-252.
23. Chou, H.T. & Kim, W. 1986. Unifying framework for version control in a CAD environment. Proceedings of the 12th International VLDB Conference, Kyoto, Japan, August 1986: 336-344.
24. Conaway, J. 1995, December. Integrated Product Development: The Technology. A white paper © Winners Consulting Group. Available at <http://www.pdmic.com/articles/jconaway.html>
25. Cross, N. 1989. *Engineering Design Methods*. John Wiley and Sons, Chichester, UK.
26. Dellen, B., Maurer, F. & Pews, G. 1997. Knowledge-based techniques to increase the flexibility of workflow management. *Data & Knowledge Engineering* 23(3): 269-295.

-
27. Dhamija, D., Koonce, D.A. & Judd, R.P. 1997. Development of a unified data meta-model for CAD-CAPP-MRP-NC verification integration. *Computers & Industrial Engineering* 33(1-2): 19-22.
 28. Dixon, J.R., Cunningham, J.J. & Simmons, M.K. 1989. Research in designing with features. Intelligent CAD I: Proceedings of IFIP TC/WG 5.2 Workshop on Intelligent CAD (edited by H. Yoshikawa and D. Gossard), Boston, MA, USA, October 1987: 137-148.
 29. Eriksson, H. 1996. Expert systems as knowledge servers. *IEEE Expert* 14: 14-19.
 30. Fagan, D.J. 1994. A blackboard approach to the integration of crankshaft analysis applications. Proceedings of the 10th IEEE Conference on Artificial Intelligence for Applications, San Antonio, Texas, USA, March 1994: 231 – 237.
 31. Fan, I.S. 2000, December. The Power of PDM. *Manufacturing Engineer*: 224-228.
 32. Fayad, M.E. & Schmidt, D.C. 1997. Object-oriented application frameworks. *Communications of the ACM* 40(10): 32-38.
 33. Feng, S. C. & Song, E.Y. 2000, November. Information Modeling on Conceptual Design Integrated with Process Planning. Recent Advances in Design for Manufacture, DE-Vol.109, Proceedings of the 2000 International Mechanical Engineering Congress and Exposition, Orlando, Florida, USA, November 2000: 123-130.
 34. Fowler, M. & Scott, K. 1997. UML distilled: applying the standard object modeling language. addison-Wesley.

-
35. Georgakopoulos, D., Hornick, M. & Sheth, A. 1995. An overview of workflow management: from process modeling to workflow automation. *Distributed and Parallel Databases* 3:119-153.
 36. Gerhard, J.F., Rosen, D., Allen, J.F. & Mistree, F. 2001. A distributed product realization environment for design and manufacturing. *Transactions of the ASME, Journal of Computing and Information Science in Engineering* 1(3): 235–244.
 37. Gray, J. & Reuter, A. 1993. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, San Mateo, California.
 38. Hanneghan, M., Merabti, M., & Colquhoun, G. 1995. The Design of an Object-Oriented Repository to Support Concurrent Engineering. Proceedings of the 1995 International Conference on Object-Oriented Information Systems (OOIS'95), Dublin, Ireland, December 1995: 200-215.
 39. Hanneghan, M., Merabti, M., & Colquhoun, G. 1998. CONCERT: A Middleware-Based Support Environment for Concurrent Engineering. Proceedings of 2nd International Symposium on Tools and Methods for Concurrent Engineering (TMCE'98), Manchester Metropolitan University, UK, April 1998: 446-455.
 40. Hardwick, M., David L., Rando, T. & Morris, K.C. 1996. Sharing manufacturing information in virtual enterprises. *Communications of the ACM* 39(2): 46-54.
 41. Hayes-Roth, B. 1985. A blackboard architecture for control. *Artificial Intelligence* 26(3): 251-321.
 42. Hayes, C.C. 1995. Flexible, Interactive Integration Architecture for Extraction of CAPP Information from CAD. Proceedings of the ASME Computer

-
- Integrated Concurrent Design Conference, September 1995, Boston, MA, USA: 825-833.
43. Heimann, P. & Westfechtel, B. 1997. A Generalized Workflow System for Mechanical Engineering. Proceedings Workshop Arbeitsplatzrechner-Integration zur Prozeßverbesserung, Aachen, Germany, Softwaretechnik-Trends. 17(3): 21-24.
44. Hillebrand, G., Krakowski, P., Lockemann, P.C. & Posselt, D. 1998. Integration-based Cooperation in Concurrent Engineering. Proceedings of the 2nd *Enterprise Distributed Object Computing Workshop (EDOC'98)*, La Jolla, CA, USA, November 1998: 344-355.
45. Hsiang, K.-K., Du, T. C.-T. & Cheng, H.-W. 1999. Applying Object-oriented database technologies in concurrent design processes. *International Journal of Computer Integrated Manufacturing* 12(3): 251-264.
46. Hsiao, W.C.D. 1990. *Feature-Based Mapping and Manufacturability Evaluation with an Open Set Feature Modeler*. Ph. D Thesis. Arizona State University.
47. ISO 10303-1. 1994. *Industrial automation systems and integration: product data representation and exchange. Part 1, Overview and fundamental principles*. International Organization for Standardization. ISO Geneva.
48. Iuliano, M. 1995. Overview of the Manufacturing Engineering Toolkit Prototype. NISTIR 5730. National Institute of Standards and Technology, Gaithersburg, MD.
49. Iuliano, M. 1997. The Role of Product Data Management in the Manufacturing Engineering ToolKit. NISTIR 6042. National Institute of Standards and Technology, Gaithersburg, MD.

-
50. Jiang, R.D., Leow, L.F., Cheok, B.T. & Nee, A.Y.C. 2000. IPD- A Knowledge-based Progressive Die Design System. Proceedings of the 5th International Conference on Computer Integrated Manufacturing, Technologies for the New Millennium Manufacturing, Singapore, March 2000:1048-1059.
 51. Jiang, R.D., Zhang, W.Z. & Cheok, B.T. 2004. Object-Oriented Feature Based Development for Progressive Dies. Proceedings of the International Conference on Scientific and Engineering Computation (IC-SEC 2004). Singapore, June 2004 (CD publication).
 52. Jeng, T.S. & Eastman, C.M. 1999. Design process management. *Computer-Aided Civil and Infrastructure Engineering* 14(1): 55-67.
 53. Johnson, R. & Foote, B. 1988. Designing Reusable Classes. *Journal of Object-Oriented Programming*, 1 (2), 22-35.
 54. Karsai, G & Gray, J. 2000, March. Design Tool Integration: an Exercise in Semantic Interoperability. Proceedings of the IEEE Engineering of Computer-based Systems, Edinburgh, UK, March 2000: 272-278.
 55. Katz, R.R., Bhateja, R., Chang, E.E.L., Gedye, D. & Trijanto, V. 1987. Design version management. *IEEE Design and Test* 14: 12-22.
 56. Katz, R.H., Chang, E. & Kahn, K.M., 1986. A Version Server for Computer-Aided Design Database. ACM/IEEE 24th Design Automation Conference, Las Vegas, NV, USA, June 1986: 27-33.
 57. Katz, R.H. & Chang, E. 1987. Managing Change in a Computer-Aided Design Database. Proceedings of the 13th International Conference on Very Large Data Bases, Brighton, GB, September 1987: 455-462.

-
58. Kim, Y., Kang, S.H., Lee, S.H. & Yoo, S.B. 2001. A distributed, open, intelligent product data management system. *International Journal of Computer Integrated Manufacturing* 14(2): 224-235.
 59. Law, H.W. & Tam, H.Y. 2000. Object-Oriented analysis and design of computer aided process planning systems. *International Journal of Computer-Integrated Manufacturing* 13(1): 40-49.
 60. Lee, Y. T. 1999. Information Modeling: From Design to Implementation. Proceedings of the Second World Manufacturing Congress, Universities of Durham, Durham, U.K., September 27-30, 1999: 315-321.
 61. Lee, I.B.H., Lim, B.S. & Nee, A.Y.C. 1993. Knowledge-based process planning system for the manufacture of progressive dies. *International Journal of Production Research* 31(2): 251-278.
 62. Leach, P. and Perry, D., 1996, CIFS: A Common Internet File System. Web publication available at: <http://www.microsoft.com/mind/1196/cifs.asp>.
 63. Li, W.D, Fuh, J.Y.H. & Wong, Y.S. 2004. An Internet-enabled integrated system for co-design and concurrent engineering. *Computers in Industry* 55 (1): 87-103.
 64. Liang, J., Shah J.J., D'Souza, R., Urban, S.D., Ayyaswamy, K, Harter, E & Bluhm, T. 1999. Synthesis of consolidated data schema for engineering analysis from multiple STEP application protocols. *Computer-Aided Design* 31 (7): 429-447.
 65. Liu, T. & Xu, X. 2001. A review of web-based product data management systems. *Computers in Industry*. 44: 251-262.
 66. Madison, E.M.D, Wilbur, G.L.T. & Wu, J.C.T. 1988. Data-driven CIM. *Computers in Mechanical Engineering*, May/June 1988:38-42.

-
67. Madni, A.M. & Madni, C.C. 1997. An adaptive wide-area design process manager for collaborative multichip module design. Proceedings of 1997 IEEE Multi-Chip Module Conference (MCMC '97), Santa Cruz, CA, February 4-5, 1997: 63-72.
 68. Maropoulos, P.G. 1995. Review of research in tooling technology, process modeling and process planning. *Computer Integrated Manufacturing Systems*, 8 (1): 13-20.
 69. Marefat, M., Malhotra, S. & Kashyap, R.L. 1993. Object-oriented intelligent computer-integrated design, process planning, and inspection. *Computer* 26(3): 54-65.
 70. McClatchey, R., Kovacs, Z., Estrella, F., Le Goff, J.-M., Chevenier, G., Baker, N., Lieunard, S., Murray, S., Le Flour, T. & Bazan, A. 1998. The integration of product data and workflow management systems in a large scale engineering database application. Proceedings of the 1998 International Database Engineering and Applications Symposium (IDEAS'98), Cardiff, Wales, U.K., July 8-10, 1998: 296–302.
 71. McFadden, F.R. 1989. Object-oriented techniques in computer integrated manufacturing. Proceedings of the Twenty-Second Annual Hawaii International Conference on System Sciences, Kailua-Kona, HI, USA, June 1989: 64-69.
 72. Megale, A., Martins, F., Sakamoto, F., Bueno, A.L. & Rodrigues, V. 1991. Using a Blackboard Architecture in CAD-CAM Systems Integration. Proceedings of the 3rd International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE '90), Charleston, SC, USA, July 1990: 131-140.

-
73. Mervyn, F., Kumar, A. S., Bok, S.H. & Nee, A.Y.C. 2003. Development of an Internet-enabled interactive fixture design system. *Computer Aided Design* 35(10): 945-957.
 74. Microsoft. 1998. *Microsoft White Paper: Windows DCOM Architecture*.
 75. Miles, J.C., Gray, W.A., Carnduff, T.W., Santoyridis, I. & Faulconbridge, A. 2000. Versioning and configuration management in design using cad and complex wrapped objects. *Artificial Intelligence in Engineering* 14(3): 249-260.
 76. Mowchenko, M. 1996. *Intelligent Independent Features: Manufacturing Features Which Ensure Their Own Manufacturability*. A PhD thesis presented to the University of Calgary.
 77. Molinari, M., Nammuni, K. & Cox, S. 2004. Integration of chargeable web services into engineering applications. Proceedings of the UK e-Science All Hands Meeting 2004, 31 August – 3 September, Nottingham, UK. Available at: <http://www.allhands.org.uk> (accessed in July 2006)
 78. Nee, A.Y.C. & Cheok, B.T. 2001. Intelligent techniques for the planning, design, and manufacture of progressive dies. in *Computer-Aided Design, Engineering, and Manufacturing: Systems Techniques and Applications* (CRC Press) Volume III: 7.1-7.27.
 79. Nii, P. H. 1996. Blackboard systems: the blackboard model of problem solving and the evolution of blackboard architectures. *AI Magazine*. Summer 1996: 38-53.
 80. Norrie C.M. 1995. Integration approaches for CIM. Proceedings of the 1995 ACM SIGMOD international conference on Management of data, San Jose, California, USA, May 1995: 470.

-
81. OMG 2002. Common Object Request Broker Architecture (CORBA/IIOP), version 3.0, formal/2002-06-01. Object Management Group.
 82. Oussalah, C & Urtado, C. 1997. Complex object versioning. *Lecture Notes in Computer Science* v1250: 259-272.
 83. Palani, R., Wagoner, R.H. & Narasimhan, K. 1994. Intelligent design environment: a knowledge-based simulation approach for sheet metal forming. *Journal of Materials Processing Technology* 45: 703-708.
 84. Park, H.J. & Yoo, S.I. 1995. Implementation of a Version Manager on an Object-Oriented Database Management System. Proceedings of the 1995 International Conference on Object Oriented Information Systems (OOIS'95), Dublin, Ireland, UK, 18-20 December 1995: 323-336.
 85. Plasil, F. & Stal, M. 1998. An architectural view of distributed objects and components in CORBA, Java RMI and COM/DCOM. *Software Concepts and Tools* 19: 14-28.
 86. Prasad, B. 1996. *Concurrent Engineering Fundamentals, Volume I: Integrated Product and Process Organization and Volume II: Integrated Product Development*. Prentice Hall PTR.
 87. Qiang, L., Zhang, Y.F. & Nee, A.Y.C. 2001. A distributive and collaborative concurrent product design system through the WWW/Internet. *The International Journal of Advanced Manufacturing Technology* 17:315-322.
 88. Ramakrishnan, R. & Janaki, R.D. 1996. Modeling design versions. Proceedings of the 22nd International conference on VLDB (VLDB'96). Mumbai (Bombay), India, September 1996: 556-566.
 89. Ramanathan, J. 1996. Process improvement and data management. *IIE Solutions* 28 (12): 24 - 27.

-
90. Ranft, M.A., Rehm, S. & Dittrich, K.R. 1990. How to share work on shared objects in design database. Proceedings of the Sixth IEEE International Conference on Data Engineering, February 5-9, 1990, Los Angeles, California, USA: 575-583
 91. Regli, W.C. 1997. Internet-enabled Computer Aided Design. *IEEE Internet Computing* 1(1): 39-50.
 92. Rehm, S., Raupp, T., Ranft, M., Langle, R., Hartig, M., Gotthard, W., Dittrich, K. & Abramowitz, K. 1988. Support for design process in a structurally object-oriented database system. In Dittrich, K.R., editor, Proc. 2nd Intern Workshop on Object-Oriented Database Systems, Bad Münster am Stein-Ebernburg, FRG, September 27-30, 1988: 80-97.
 93. Roller, D. & Eck, O. 1999. Knowledge based techniques for product database. *International Journal of Vehicle Design* 21(2/3): 243-265.
 94. Roller, D., Eck, O. & Dalakakis, S. 2002a. Integrated version and transaction group model for shared engineering databases. *Data & Knowledge Engineering* 42: 223-245.
 95. Roller, D., Eck, O. & Dalakakis, S. 2002b. Advanced database approach for cooperative product design. *Journal of Engineering Design* 13(1): 49-61.
 96. Roy, U., Bharadwaj, B., Chavan, A. & Mohan, C.K. 1995. Development of a feature based expert manufacturing process planner. Proceedings of the 1995 IEEE 7th International Conference on Tools with Artificial Intelligence, Herndon, VA, USA, November 05 - 08, 1995: 63-70.
 97. Rundensteiner, E.A. 1993. Design tool integration using object-oriented database views. Proceedings of the 1993 IEEE/ACM International Conference on Computer-Aided Design, Santa Clara, CA, USA, November 1993: 104-107.

-
98. Schrmann, B. & Altmeyer, J. 1997. Modeling design tasks and tools - the link between product and flow model. Proceedings of the 34th ACM/IEEE Design Automation Conference, Anaheim, CA, USA, June 1997: 564-569.
 99. Schwartz, J. & Westfechtel, B. 1993. Integrated data management in a heterogenous CIM environment. Proceedings of the 7th IEEE Annual European Computer Conference (COMPEURO 93): Computers in Design, Manufacturing, and Production, Paris, France, 24-27 May 1993: 248-257.
 100. Shah, J.J., Dedhia, H., Pherwani, V. & Solkhan, S. 1997. Dynamic interfacing of applications to geometric modeling services via modeler neutral protocol. *Computer-Aided Design* 29 (12): 811-824.
 101. Shah, J.J. 1988. Feature transformations between application-specific feature spaces. *Computer-Aided Engineering Journal* 5(6):247-255.
 102. Shah, J.J. & Urban, S.D. 1998, September. *Integrated product design environment. DARPA-RaDEO Final Report*. ASU Design Automation lab.
 103. Singh, N. 1996. *System Approach to Computer-Integrated Design and Manufacturing*. New York: Wiley.
 104. Srihari, K., Amal Cecil, Joe & Emerson, C.R. 1994. Blackboard-based process planning system for the surface mount manufacture of PCBs. *International Journal of Advanced Manufacturing Technology* 9(3): 188-194.
 105. Sriram, D. & Logcher, R. 1993. The MIT Dice project. *IEEE Computer* 26 (1): 64 – 65.
 106. Sun. 2002. *Java 2 platform Enterprise Edition Specification v1.4*.
 107. Tamine, O. & Dillmann, R. 2003. KaViDo—A web-based system for collaborative research and development processes. *Computers in Industry* 52(1): 29-45.

-
108. ten Bosch, K. O., van der Wolf, P. & Bingley, P. 1993. Flow-based user interface for efficient execution of the design cycle. Proceedings of the 1993 IEEE/ACM International Conference on CAD, Santa Clara, CA, USA, November 1993: 356-363.
109. Teti, R. & Kumara, S.R.T. 1997. Intelligent computing methods for manufacturing systems. *CIRP Annuals—Manufacturing Technology* 46(2):629-652.
110. Thomas, K. K. & Fischer, W. G. 1996. Integrating CAD/CAM software for process planning applications. *Journal of Materials Processing Technology* 61 (1996): 87-92.
111. Tian, G.Y., Yin, G.F. & Taylor, D. 2002. Internet-based manufacturing: a review and a new infrastructure for distributed intelligent manufacturing. *Journal of Intelligent Manufacturing* 13(5):323-338
112. Tor, S.B., Britton, G.A., & Zhang, W.Y. 2003. Indexing and retrieval in metal stamping die design using case-based reasoning. *Journal of Computing and Information Science in Engineering* 3: 353-362.
113. Turgut, D., Aydin, N., Elmasri, R. & Turgut, B. 2001. Utilizing object-oriented databases for concurrency control in virtual environments. Proceedings of the 25th International Computer Software and Applications Conference (COMPSAC 2001), Invigorating Software Development, Chicago, IL, USA, 8-12 October 2001: 409-414.
114. Urban, S.D., Ayyaswamy, K., Fu, L., Shah, J., Liang, J. 1999a. Integrated product data environment: data sharing across diverse engineering applications. *International Journal of Computer Integrated Manufacturing* 12 (6): 525-540.

-
115. Urban, S.D., Dietrich, S.W., Saxena, A. & Sundermier, A. 2001. Interconnection of distributed components: an overview of current middleware solutions. *Journal of Computing and Information Science in Engineering*. 1: 23-31, ASME.
 116. Urban, S.D., Fu, L. & Shah, J.J. 1999b. The implementation and evaluation of the use of CORBA in an engineering design application. *Software-Practice & Experience* 29 (14): 1313-1338.
 117. Urban, S.D., Shah J.J., Liu, H. & Rogers, M. 1996. The shared design manager: Interoperability in engineering design. *Integrated Computer-Aided Engineering* 3 (3): 158-177.
 118. Urban, S.D., Tjahjadi, M. & Shah, J.J. 2000. A case study in mapping conceptual designs to object-relational schemas. *Concurrency-Practice and Experience* 12 (9): 863-907.
 119. U.S. Air Force. 1981. *Integrated Computer-Aided Manufacturing (ICAM) Architecture*. Part II. Material Laboratory, U.S. Air Force Wright Aeronautical Laboratories.
 120. Wampler, B. 2001. *The essence of Object-Oriented Programming with Java™ and UML*. Addison Wesley Professional.
 121. Wang, G.R., Yu, G., Zhou, Y.F., Shan, J.D. & Zheng, H.Y. 1993. DODBMS/CIM: A distributed object-oriented database management system for CIM applications. Proceedings of the 10th IEEE Region Conference on Computer, Communication, Control and Power Engineering, Beijing, China 19-21 October 1993: 303-306.
 122. Wang, G.X, Zhang, W.Z, Lu, C., Nee, A.Y.C. 2004. A distributed, persistent and transactional cache for knowledge-based engineering, Proceedings of the

-
- International Conference on Scientific and Engineering Computation (IC-SEC-2004) (CD publication), Singapore, June 2004.
123. Wang, H.F. & Zhang, Y.L. 2002. CAD/CAM integrated system in collaborative development environment. *Robotics and Computer Integrated Manufacturing*, 18: 135–145.
124. Westfechtel, B. 1996. Integrated product and process management for engineering design applications. *Integrated Computer-Aided Engineering*, 3 (1): 20-35.
125. Westfechtel, B. 2000. *Models and Tools for Managing Development Processes*. Berlin; New York: Springer-Verlag.
126. Wolf, P.V.D. 1994. *CAD Frameworks: Principles and Architecture*. Dordrecht: Kluwer Academic, Boston.
127. Wong, T.N. & Leung, C.B. 1995. Feature conversion between neutral features and application features. *Computers & Industrial Engineering*, 29(1-4): 625-629.
128. Wong, T.N. & Leung, C.B. 2000. An Object-Oriented Neutral Feature Model for Feature Conversion. *International Journal of Production Research*, 38(15): 3573-3601.
129. Wu, T., Xie, N. & Blackhurst, J. 2004. Design and implementation of a distributed information system for collaborative product development. *Journal of Computing and Information Science in Engineering* 4(4): 281-293.
130. Xie, S.Q., Tu, P.L., Aitchison, D., Dunlop, R. & Zhou, Z. D. 2001. A WWW-based integrated product development platform for sheet metal parts intelligent concurrent design and manufacturing. *International Journal of Production Research* 39(17): 3829-3852.

-
131. Xu, X. & Liu, T. 2003. A Web-enabled PDM system in collaborative design environment. *Robotics and Computer-Integrated Manufacturing* 19: 315-328.
 132. Yoon, D.H. & Shaikh, F. Z. 2000. Integrating CAD and CAM with CORBA. Proceedings 7th IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2000), Edinburgh, UK, April 2000: 3-8.
 133. Zha, X. F. & Du, H. 2002. Web-based collaborative framework and environment for designing and building robotic systems. Proceedings of the 2002 IEEE International Conference on Robotics and Automation (ICRA 2002), Washington, DC, USA, May 11-15, 2002: 2196-2201.
 134. Zha, X.F., Sriram, R.D. & Lu, W.F. 2003. Knowledge intensive collaborative decision support for design process. Proceedings of DETC'03, ASME 2003 Design Engineering Technical Conferences and Computers and Information in Engineering Conference, Chicago, IL USA, 2-6 September 2003: 425-438.
 135. Zhang, W.J. & Luttermelt, C.A. 1995. On the Support of Design Process Management in Integrated Design Environment. *CIRP Annals - Manufacturing Technology* 44(1): 105-108.
 136. Zhang, W.Z., Jiang, R.D., Cheok, B.T. & Nee, A.Y.C. 2002. An innovative and practical design automation system for progressive dies. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture* 216(12): 1611-1619.
 137. Zhang, Y.P., Zhang, C. & Wang, H.P. 2000. Internet based STEP data exchange framework for virtual enterprises. *Computers in Industry* 41(1): 51-63.

138. Zimmermann, J.U., Haasis, S., & Van Houten, F.J.A.M. 2002. ULEO-Universal Linking of Engineering Objects. *CIRP Annals - Manufacturing Technology* 51(1): 99-102.

PUBLICATIONS FROM THIS RESEARCH

- Journal paper: Zhang WZ, Wang GX, Cheok BT, Nee AYC. **A Functional Approach for Standard Component Reuse**, *International Journal of Advanced Manufacturing Technology* Volume 22, Issue 1-2, 2003, Pages 141-149.
- Conference paper: Wang GX, Zhang WZ, Nee AYC. **Virtual Knowledge Repository for Intelligent and Distributed Feature-driven Product Realization**. Presented in the conference of 34th International MATADOR Conference, 7 – 9 July 2004, Manchester, UK.
- Conference paper: Zhang WZ, Wang GX, Lu C, Nee AYC. **An Agent-based Organization of Web Services in a Computational Grid**, Presented in the International Conference on Scientific and Engineering Computation (IC-SEC 2004), 30 June – 2 July, Singapore.
- Conference paper: Wang GX, Zhang WZ, Lu C, Nee AYC. **A Distributed, Persistent and Transactional Cache for Knowledge-based Engineering**, Presented in the International Conference on Scientific and Engineering Computation (IC-SEC 2004), 30 June – 2 July, Singapore.
- Journal paper: Zhang WZ, Wang GX, Lu C, Nee AYC. **A Staged Approach for Feature Extraction from Sheet Metal Part Models**, *International Journal of Production Research*, in press.
- Journal paper: Wang GX, Zhang WZ, Nee AYC. **An Integration Framework for Digital Progressive Die Design and Manufacturing**, *Journal of Wuhan University of Technology*, in press.