**Stellingen** (proposition in English)

1. The work described in this thesis was interpreted as "... solving complex non-linear formulas with database." in the "Quality Assessment of Research" (VSNU Report, 1994). This interpretation is nonsense in all aspects.

2. The object-orientation promotes a decentralised manner of organising a large and complex software system. Individual system components (or subsystems) are made as encapsulated objects. Communications among subsystems are only by means of message passing. Message passing, by definition, will not influence any constituent elements of subsystems. The advantage of decentralised manner of organisation makes possible the concurrent development of individual subsystems, and it is being appreciated nowadays. This recognition seems relevant to the following fashionable notions: distributed system (in the technical domain), democracy (in the political domain), market-oriented economy (in the economic domain), etc..

3. Information relativity must be a universal principle in the information system development. It is a key to achieving a dynamic structure of a knowledge representation. The dynamic structure is very important in modelling knowledge, in which the boundary between schema and instance is impossibly clear, see also Lockerman et al. (1992). (The information relativity principle is described in Appendix A of this thesis.)

4. The comment made by, among others, Tomiyama & ten Hagen (1987a) and Blessing (1993), that an intelligent CAD system must be process-oriented, can be argued. In the first place, the way that a CAD system is developed as such, processes are around a product model, does not mean that the system can only handle geometric information, because nowadays a product model contains more than product geometrical information. In the second place, the product-orientation does not exclude process management. Modern integrated CAD/CAM systems, e.g. CIMOME must explicitly represent both product data and process data in the computer. (Chapter 2 of this thesis.)

5. "Description of mating conditions does not make explicit the kinematic behaviour of a mechanism to support a kinematics analysis package. Therefore, if future CAE systems are to share product information, they must support descriptions of function and behaviour as well as mating conditions." Henson B.W. et al. (1994) in Computer Aided Conceptual Design, J. Sharpe and V. Oh (eds), page 95-112.

The above statement actually neglects the structural nature of information at the kinematic design stage of machines. The information at this design stage is structural and usually represented by the human designer with the kinematic mechanism diagram with design notes. GMM is an explicit data representation of this structural information, so the computer system based on GMM will well support the conceptual design of machines. (Chapter 5 of this thesis.)

6. Removal of the condition set on the application of the object relativity principle by Smith & Simth (1978), *if an object A is viewed as an attribute of an object B, the functional correspondence (from B to A) must be n:1*, has been neglected in the conceptual data modelling, sometimes. (Chapter 3 of this thesis.)
7. The methods in the literature for modelling composite objects are rooted in the data abstraction of aggregation (or has-a link), which are only applicable to the situation where types and/or numbers of element objects are known to the database schema designer. Introducing the grouping data abstraction to the composite object representation, which is first described in this thesis, provides an opportunity to model composite objects more dynamically. (Chapter 3 of this thesis.)

8. The features of intelligent CAD/CAM systems are: (1) hierarchical system organisation and (2) model-based reasoning. (Chapter 10 of this thesis.)

9. Object-oriented data modelling does not guarantee that an appropriate application data model can be constructed. Object-oriented data modelling, thought powerful in modelling expressiveness, is just a hammer (with respect to an operator) or a pen-brush (with respect to a painter). A hammer and the use of a hammer are two completely different things. GMM developed in this thesis is unique in its way of using the semantic data modelling, based on a thorough understanding of the application, i.e. mechanism design.

10. In the technical domain, the term integration is overwhelming, as we have seen, tool integration, process integration, framework integration, interface integration, to name but a few. However, in the human society, integration may not always be appreciated. An obvious limitation of integration is the decreasing of the "freedom" of each integrated element. This, together with the increasing degree of the intelligence of the computer system, may make us worry about the potential danger of the occurrence of a war in the computer world.

11. The relationship between authors and publishers should be "many-to-many". Therefore both parties should have the right to choose their potential partner simultaneously. The modern concurrent engineering principle will eventually be applied to this yet unnoticeable domain so that new ideas can be published sooner and to the mutual satisfaction.

12. The most valuable idea won't be accepted for the first time, with a full agreement of referees. The larger differences between referees, the higher value the idea.

13. Data modelling has something to do with art, such as painting. In painting, the real world is depicted with pen-brushes and sheet, while in data modelling the real world is depicted with data structures and computer. Of course, a data modelling expert should not become an expressionist, but a realist.

14. 'Enjoy life' is defined as having fewer constraints imposed on life.

15. It is more difficult to say 'No' than 'Yes'.

16. If 'emancipation is accepted there should be no difference between she and he.

W. Zhang
at Delft
2 Oct. 1994
An Integrated Environment for CAD/CAM of Mechanical Systems
An Integrated Environment for CAD/CAM of Mechanical Systems

Proefschrift

ter verkrijging van de graad van doctor aan de Technische Universiteit Delft, op gezag van de Rector Magnificus, Prof.ir. K.F. Wakker, in het openbaar te verdedigen ten overstaan van een commissie aangewezen door het College van Dekanen op dinsdag 15 november 1994 te 10:30

door

Wenjun ZHANG

master of science in mechanical engineering

geboren te Shanghai City, China
An Integrated Environment for CAD/CAM of Mechanical Systems / Wenjun ZHANG. - Delft: Delft University of Technology. - With ref.

ISBN 90-370-0113-0

Subject headings: CAD/CAM/Environment/Mechanical System/Design Automation

Copyright © 1994 by Wenjun ZHANG

All rights reserved. No part of this thesis may be reproduced or transmitted in any form or by any means, electronic, mechanical, photocopying, any information storage and retrieval system, or otherwise, without written permission from the copyright owner.
To my wife Xiaowei and my son Meng
To my parents
# Contents

## Preface

## Abbreviations

1 **Introduction**
   1.1 The subject of the study 1
   1.2 An Analysis of the Related Work 3
   1.3 The Main Philosophy behind the CIMOME Development 6
   1.4 The Scope of the Work 8
   1.5 The Organization of the Thesis 9

2 **Design of the CIMOME Architecture**
   2.1 Introduction 13
   2.2 The Mechanism Design Process 14
   2.3 Design Principles for the Architecture of CIMOME 18
   2.4 The CIMOME Architecture 27
   2.5 Implementation Consideration 34
   2.6 Discussion and Conclusions 35

3 **Data Modelling: an approach to CIMOME**
   3.1 Introduction 39
   3.2 Definition and Discussion of Basic Concepts of Database 39
   3.3 A Hybrid Approach to Developing Conceptual Database Schema 42
   3.4 The Database Approach to the CIMOME Development 58
   3.5 Features of ABC system 60
   3.6 Discussion and Conclusions 62

4 **The Mechanism Development Model**
   4.1 Introduction 63
   4.2 Model 64
   4.3 A General Framework of the Product Model 65
   4.4 A Partial Requirement Model for Mechanism Design 69
   4.5 A Partial Process Model for Mechanism Design 74
   4.6 Discussion and Conclusions 78

5 **A Database Model for the Conceptual Structure of Mechanisms** 81
5.1 Introduction 81  
5.2 The Need of GMM for CIMOME 82  
5.3 Requirements on and Philosophy behind Developing GMM 87  
5.4 An Object-based Description of Mechanisms 89  
5.5 Conceptual Schema of GMM 92  
5.6 Link between GMM and the Machine Assembly Model 108  
5.7 Implementation and Preliminary Verification of GMM in Use 112  
5.8 Discussion and Conclusions 113  

6  The Computer Aided Mechanism Identification 123  
6.1 Introduction 123  
6.2 Identification of Mechanism Structures on Different Abstraction Layers 124  
6.3 An approach to the Mechanism Identification 126  
6.4 Algorithms 136  
6.5 Discussion and Conclusions 141  

7  Editors in CIMOME 145  
7.1 Introduction 145  
7.2 A Requirement Analysis of the EDMECH System 146  
7.3 The Architecture of the EDMECH System 148  
7.4 Implementation 153  
7.5 Discussion and Conclusions 158  

8  Automating Finite Element Modelling of Mechanisms 163  
8.1 Introduction 163  
8.2 Delft-FEM: Review and Classification 164  
8.3 General Solution to Automating Finite Element Modelling 173  
8.4 A Finite Element Mechanism Model 178  
8.5 Reasoning Strategy and Reasoning: an outline 183  
8.6 System Demonstration through Cases 189  
8.7 Discussion and Conclusions 195  

9  Design Synthesis Tools in CIMOME: 197  
Implementation Overview and Perspective 197  
9.1 Introduction 197  
9.2 CAMDES: a Sub-Environment for Compound Cam Mechanism Design 198  
9.3 DSYNOP: a Sub-Environment for Dimension Synthesis via Optimization 218  
9.4 LINKDES: a Sub-Environment for Type and Dimension Synthesis 223  
9.5 Discussion and Conclusions 227  

10 Concluding Remarks and Future Work 229  
10.1 Introduction 229  

(ii)
10.2 Concluding remarks 229
10.3 Future work 234

Reference 237

Summary 249

Samenvatting 255

Appendix A. Definition of Concepts from Information Technology 261

Curriculum Vitae 263
Preface

This thesis describes my major work in the CIMOM project carried out across the Laboratory of Mechanization of Production and the Laboratory of Engineering Design and CAD, in the Delft University of Technology, the Netherlands, during 1991-1994.

My promotors, Prof.ir. H.A. Crone and Prof.dr.ir. K. van der Werff, have made great contributions to my work. I am very grateful to Prof.ir. H.A. Crone for his directing and backing, and to Prof.dr.ir. K. van der Werff for his leading me to the data and knowledge approach to the modern CAD of machines.

The work would not be successfully carried out without the coordination of Ir. A. van Dijk, the manager of the Design and Production Division, who also helps me to translate the summary of my thesis into the Dutch language. I owe Dr.Ir. A.J. Klein Breteler a lot. He first introduced me as a visiting researcher to the Laboratory of Mechanization of Production in 1990. His interest in my work has never been stopped ever since then. His meticulous and critical reading of the concept version of this thesis leads to further improvement of my work.

My family and I will never forget friendship with and help from Ing. J. Stout (coordinator of the Laboratory) and E.M. Koets-Stoutjesdijk (secretary of the Laboratory). W. van der Bos (technician of the Laboratory) helps to manufacture the model of a mechanism with the constant force transmission property, which I discovered in 1990 and now use as part of the cover of this thesis.

A lot of former graduate students were involved in the CIMOM project, such as Ir. T. Jongsma, Ir. P. van Balleghooijen, Ir. J.M. van Veen, Ir. L.C. van Wagensveld, Ir. F.A.J. Stokkermans and Ir. J. 't Hart. Working with them reminded me of the joyful days when I worked as a student tutor in the China Textile University.

I am grateful to Ir. C.A. van Lutterveld for his recommendation of my work to the well-known CIRP community. Discussion with Ir. C.M. Kalk-Kalkman on the issue of the mechanism identification helps to formulate a new approach to the mechanism identification, which is described in Chapter 6. I am also indebted to Ir. J.W. Poort and Ing. Tj. Jongeling for their help with the management of the computer system, and to Ing. J.B.W. Hoebeek for his advice concerning the program/program interface on the MS-DOS platform.

Since the project is multi-disciplinary in nature, I have had the pleasure of having discussions with researchers in other faculties of the University. I particularly appreciate the fruitful discussions with Prof.dr.ir. W. Gerhardt, Prof.dr. D.J. Mcconalogue, Dr.ir. J.H. ter Bekke,
Dr. H.K. Li and Y. Liu (the Faculty of Computer Science) on such issues as database and software engineering, and with Dr. J.S.M. Vergeest (the Faculty of Industrial Design) on the issue of product modelling with EXPRESS.

If possible, I would like to place my wife, X.W. Meng, as the second author of the thesis, not only because of her patience and understanding during the past four years but also because of her time spent and effort made in assisting me to complete my work and thesis. I owe much to my son, M. Zhang, with whom I could not afford to spend much time together in the evenings and at weekends.
Abbreviations

ABC: A main memory database management system developed in this study.
Ada: A commercial programming language.
AUTOCAD: A commercial computer-aided drawing system.
C: A commercial programming language.
CAMDES: A sub-environment of CIMOME for the compound cam mechanism design.
CASE: Computer-Aided Software Engineering.
CIMOME: An environment system for computer integrated manufacture of machines.
DBMS: DataBase Management System.
EDFUNC: A computer program system developed in this study for editing behaviour functions that a mechanism may fulfil.
EDMECH: A computer program system developed in this study for editing the conceptual structure of mechanisms.
EMBDES: A computer program system developed in this study for the preliminary layout design.
E-R: Entity Relationship model.
DSYNOP: A sub-environment of CIMOME for the dimension synthesis of mechanisms.
DYNANA: A sub-environment of CIMOME for the dynamic analysis and simulation of mechanisms.
FEM: The finite element method applied to the analysis of mechanical behaviour of a machine and developed at Delft University of Technology.
FORTRAN: A commercial programming language.
GMM: Generic Mechanism Model - a database model for the conceptual structure of mechanisms.
GFM: Generic Function Model - a database model for behaviour functions of mechanisms.
KINANA: A sub-environment of CIMOME for the kinematic analysis and simulation of mechanisms.
LINKDES: A sub-environment of CIMOME for the type and dimension synthesis of mechanisms.
MECHID: A computer program system developed in this study for the identification of mechanisms based on topology.
MPAM: Motion Property Adjacency Matrix proposed in this study.
MPG: Motion Property Graph proposed in this study.
MQ: Motion Quantity.
MTL: Mechanism Topology Level.
MUIMS: Mini User Interface Management System developed in this study.
OPLAM: A computer program system for the dimension synthesis via optimization.
ORACLE: A commercial relational database management system.
PASCAL: A commercial programming language.
PLANAR: A computer program system for the kinematic and dynamic analysis of planar mechanisms based on the finite element method.
ProEngineer: A commercial package for CAD/CAE.
RUNMEC: A computer program system for the kinematic and dynamic analysis of planar mechanisms based on the finite element method.
SAM: A planar mechanism simulation and analysis package based on the finite element method.
SmallTalk: An object-oriented programming language.
SPACAR: A computer program system for the kinematic and dynamic analysis of spatial mechanisms based on the finite element method.
TADSOL: A computer program system for the type and dimension synthesis of mechanisms.
TADSOCE: A computer program system for the cam mechanism design.
UIMS: User Interface Management System.
Introduction

This dissertation describes the development of CIMOME, which is an acronym of Computer-Integrated Manufacture of Mechanisms Environment. To formulate the subject of the study, Section 1.1 answers the following questions:

- What is CIMOME about?
- Why is it needed?

To highlight the significance of the study and derive the main philosophy behind the CIMOME development, Section 1.2 presents an analysis of the related work. Section 1.3 presents the main philosophy behind the CIMOME development. Section 1.4 describes the key issues which will be focused in this dissertation. Section 1.5 provides a brief introduction to the subsequent chapters and their relationships.

1.1 The Subject of the Study

A machine is a mechanical & electric system that performs a specific task, such as the reforming of material, transfer of information and transformation of motion and force. In the machine a part which performs the motion and force transfer is called mechanism. A mechanism is physically composed of machine components, such as a bar, a gear, a cam and a wheel. A mechanism is driven by one or more actuators, which perform the input motion to be transferred by the mechanism. In machine design, mechanisms have a significant impact on the total machine concept.

Development of mechanisms is to obey the general rules for product development. Design and manufacturing of mechanisms have long been of interest to researchers. Many studies on mechanism design have been published; among others, Van der Werff (1977) described a
general theory to use the finite element concept for the kinematic and dynamic analysis of mechanisms, Klein Breteler (1987) described a general theory to use the finite element concept for the kinematic optimization of mechanisms, Zhang (1993d,e) studied the quality of motion transfer and Kota & Chiu (1992) developed a computational approach to mapping from the design function space to the attribute space for mechanical systems. All the work mentioned above aims to contribute in finding the best solution (i.e. mechanism) for a given machine design task. However, application of these results in industrial design practice is hindered. Because on one hand, the use of these theories requires understanding of advanced mathematics and mechanics, while on the other hand, the costs for training the human designer to meet this requirement are a barrier to match the benefit resulting from the improvement of the quality of a mechanism design, especially for small and medium companies.

The advent of high-speed and low-cost computers since the 1970s provides an opportunity for researchers to develop computer-based tools to facilitate the use of the design theories in design practice. These tools chunk some knowledge for design activities in the computer so that the designer - the user of these tools, is relieved from knowing details of the theories. However, these tools are not yet widely used because of the following reasons:

- the user-friendliness of these tools remains to be improved (Zhang & Van Dijk, 1994b).
- the degree of design automation enabled by these tools remains to be increased (Zhang & Van der Werff, 1993c).
- using these tools in a coherent manner remains to be solved (Klein Breteler & Van der Werff, 1989; Zhang et al., 1993b).

The above observations initiate the present study which is a part of the project called Computer Integrated Manufacturing of Machines (CIMOM for short) at the Laboratory of Mechanization of Production. CIMOM is aimed at integrating those tools covering the whole life cycle of the machine development. The subject of the present study in particular is to develop theories, methods and a prototype system to show that this aim can be reached. The implementation in the present study is focused on the tools developed in the past at our laboratory for the conceptual design stage of the machine, the mechanism in particular. It is also envisaged that the degree of the user-friendliness and the design automation of the tools can be increased, and eventually the accessibility and usability of the system in industrial design practices will tremendously be improved.

The integration of tools means to improve communications among the tools itself as well as communications of the tools with the human designers. Communication means pass news and information to and from (Oxford Dictionary, 1976); a medium is needed. Such a medium is called, in the present study, environment, which is similar to that used in software engineering (Buxton & Druffel, 1984). So the system to be developed in this study is termed
Computer Integrated Manufacturing of Machines Environment (CIMOME for short).

1.2 An Analysis of the Related Work

1.2.1 CAD/CAM for mechanisms
Until now much attention was paid on developing packages for the analysis of the motion behaviour of mechanisms. Among many others, Sheth & Uicker (1972) developed a system based on their local coordinate system theory and the closed loop matrix approach (Sheth & Uicker, 1971). Van der Werff (1977) developed the PLANAR system based on his finite element approach. Hollars & Rosenthal (1991) reported the idea behind the APPLIED MOTION system which is based on Kane’s formulation of the motion equation for multi-body systems. Ricci (1993) reported the SPACEBAR system which takes the pivoted four bar linkage as building block to construct mechanisms. Keil & Myklebust (1984) commented that these packages still require a substantial amount of work on the part of the user in activities before the design is completed. The observation is further made by Zhang & Van der Werff (1993a) that the user-friendliness of these systems is essentially hindered because the mechanism model underlying these systems is strongly influenced by the theories for formulating the motion equations of mechanisms, e.g. the finite element approach (Van der Werff, 1977). Any user-interface system directly based on these models requires from the end-user (the designer) some knowledge of these theories.

In the synthesis of mechanisms, Rankers et al. (1976) reported the TADSOL system for Type And Dimension Synthesis Of Link mechanism employing FOURIER series. User interfaces have been built upon this package by Kaper (1987) and Leenders (1991). As stand-alone packages they are fairly successful, but integration of them to the package for analysis causes some problems. For example, the mechanism model developed by Kaper (1987) is generic in the sense that it is a template and individual mechanisms are merely instances of this template, but the model is based on the finite element approach to formulation of the motion equations of mechanisms. Klein Breteler (1987) reported the OPLAM system for the dimension synthesis based on the combination of the finite element approach and the optimization technique. The package is of generality to most types of planar mechanisms but requires the end-user to have some knowledge of the finite element approach for optimization purpose. Kramer & Barrow (1989) developed a package for path generator which is more user-friendly because the optimization technique used in the package is hidden from the end-user. The type of mechanism used for path generator in this package is only the pivoted four bar mechanism. They are free from the problem of the mechanism model, because only one type of mechanisms is considered.

There are some reports on integration of several mechanism design tools. Most of them were concerned with the integration of analysis systems and geometric solid modelling systems.
Keil & Myklebust (1984) developed a system which integrates a kinematic analysis system, an automatic solid model generation system and a 3D animation system. Shah et al. (1994) described a system which integrates a kinematic and dynamic system, an automatic finite element mesh generation system and a finite element analysis system. The integration in these systems usually follows a pattern input-output, see Fig.1.1. In this pattern a design process always starts from the beginning and sequentially forwards till the end. High priority is put on the input/output data. Their connections are sometimes automated, but the semantics of these data with respect to the mechanism structures are left to the end-user. The research in the direction of the computer-aided management of the design process, iterative process in particular, is seldom done. In the view point of systems development, they are somewhat function oriented.

Another observation regarding integration is described in the following. The mechanism design can globally be divided into the conceptual design, embodiment design and detail design, see Fig.1.2. Design tools can then be classified in terms of these design stages, see Fig.1.3. A concept, termed horizontal integration and vertical integration, is introduced. The horizontal integration is meant for the communication of two tools functioning at one design stage while the vertical integration for the communication of two tools functioning at two different design stages, respectively. As opposed to this concept the previously mentioned
work on integration of tools for mechanism design is no more than performed at the vertical integration.

![Diagram](image)

*Figure 1.2. General Design Process for Mechanisms with respect to the Models*

![Diagram](image)

*Figure 1.3. Horizontal Integration versus Vertical Integration*

1.2.2 CAD/CAM for general products

The research on the integration of CAD/CAM for a general product was started in the early 1980s. Kimura et al. (1984) described the architecture of their integrated system and the representation system for a designed object. The concept called *product model* was proposed
and its role in the integration of CAD/CAM for products was described. The data representation formalism for a designed object is relatively \textit{ad hoc}, and was focused on the end result of a design process, which is considered inappropriate for supporting the conceptual design (Tommiyam & Ten Hagen, 1987a,b). The role of modelling the product structure in CIM was further justified by Lindberg & Agerman (1992), in which a general model for product structures was proposed, based on the well known E-R (Entity-Relationship) formalism. It is observed that the model is merely suitable to the end result of the product. The capability of representing the machine as a product for the conceptual design has yet to be justified. On one hand the product model is recognized to be critical on integration of CAD/CAM, but on the other hand some confusions have been found on the concept of the product model, its content and its relationship with the \textit{process model}. For example, recently researchers argue that CAD system must be process-oriented (Blessing, 1993).

1.2.3 Software systems development

The environment concept has been studied in systems development for more than a decade. The initiative of the study is to automate the building of an application software system. The idea is to put all tools for systems development, such as editor, compiler, linker and debugger into a common platform by facilitating the communication among these tools. Such a common platform is termed as \textit{software environment} or \textit{environment}. The key solution to building such an environment is to set up a common database in which the input/output data of each tool concerning an application program is stored. This is termed \textit{common database} approach (Buxton & Druffel, 1984). The philosophy behind this approach is in the following:

- The key to automation in systems development is systematically to capture and structure various \textit{objects} that are produced and consumed in a design process: requirement specifications, design specifications, software modules, etc.
- The state of a project is then defined in terms of the state of the objects that have been produced at a given point of time. The progress of the development process is reflected in terms of the growth of the objects in an information bases, see Fig.1.4.

1.3 The Main Philosophy behind the CIMOME Development

Taking the discussion of the preceding two sections as input we shall set up some philosophies or strategies behind the CIMOME development and conclude them as follows:

- \textit{Learn from software systems development}. By making the analogy between mechanisms and software systems, the environment concept is equally applied to our application context - the mechanism development. The common database approach will be adopted in the CIMOME development. The environment concept with the common database approach should overcome the shortcoming of the integration pattern input-output (see Section 1.2.1) and promote an iterative or a non-sequential design process.
Learn from the human society. The term integration is certainly not new to the human society. It is observed from the human society that in any types of integration there exists a question called degree of integration, which may further be measured in terms of several dimensions, integration dimension. The common database approach to tools integration only concerns data dimension. More integration dimensions need to be considered in CIMOME, see later the discussion in Chapter 2.

Place emphasis on the mechanism structure model in the computer. Such a model must be defined neutral in the sense that any theory for the formulation of the motion equations of the mechanism should not influence the model (see later Chapter 5). A neutral model of mechanism structures will fundamentally improve the user-friendliness of a design support system for the design artifact specification. (The increase of the user-friendliness of computer-based tools is one of the goals of this study, see Section 1.1.) The presence of such a model will also facilitate the design automation as shown in Fig.1.5, where the reduction of the end-user's activities is implied.
- Organize the mechanism structure model into layers corresponding to the general design process. We shall define the mechanism structure model into three layers: conceptual, embodiment and detail (Fig.1.2), instead of the last two layers only. It is the key to the support for the conceptual design that the mechanism structure model has an explicit layer corresponding to the conceptual design stage. With respect to each layer there is a set of tools for the human designer to use; the integration shall be performed both horizontally and vertically, see Fig.1.3.

- Exploit database or data modelling technology. There are actually two functions of database modelling technology: one is for the management of a large amount of data and the other is for the representation of application semantics. It is interesting to find that the first function of database modelling technology is more highlighted in the existing engineering applications, but we shall concentrate on its second function in the CIMOME development.

- Use the existing computer resources developed at our group if possibly at the implementation of CIMOME. This is intended to have a fast and economical prototype of CIMOME, which can further be polished to be used for small and medium companies.

1.4 The Scope of the Study

The study described in this dissertation is concentrated in the following aspects:

- to develop theories for an integrated environment in the mechanical engineering domain, e.g. design principles of the environment architecture, etc.

- to develop methods and implementation techniques for CIMOME. This includes the study of potential applications of data modelling in the CIMOME development.

- to study the product modelling theory with respect to the CIMOME development. This shall cover both the concepts of product model and process model.

- to develop a data representation for conceptual mechanism structures. The model must be implemented to show its roles in

  - integration of tools,
  - improvement of the user-friendliness of tools in CIMOME, and
  - improvement of the design automation enabled by these tools.

- to implement prototype system of CIMOME which covers these tools (for the mechanism design in particular) developed in the past at our laboratory. The prototype system is used to verify the theories, methods and implementation techniques and show perspective of a polished CIMOME in future.
1.5 The Organization of the Dissertation

Chapter 2 presents some concepts underlying the CIMOME architecture development, such as integration dimensions - an integration model for CIMOME (data integration, process integration and user-interface integration), criteria for a better architecture of an integrated environment, and environment architecture design principles. The CIMOME architecture is developed based upon these concepts. The implementation strategy is described with consideration of the characteristics of those mechanism design tools to be integrated.

Chapter 3 introduces the data or database modelling technology to the CIMOME development. The justification of using database technology as a key to developing CIMOME is addressed. The emphasis is placed on the database approach to the conceptual data model of applications. Based on an overview of the current data modelling method with regard to the needs of machine design, a hybrid conceptual data modelling approach is found necessary for the CIMOME development. This hybrid approach takes known data abstractions together with a set of fundamental modelling concepts and guidelines. The approach with its representation schemata is used throughout the subsequent chapters. Furthermore, arguments are presented to use a main memory database management system for the CIMOME development in particular. The development of a main memory database management system is briefly discussed. This chapter describes a tool for both the CIMOME system design and the implementation.

Chapter 4 presents a framework of the mechanism development model by following the general product model theory. The chapter starts with the discussion of the model concept and based on this discussion the notion of the product model as well as the process model is clarified. A general framework of the product model is proposed, which categorizes the product information into object and object's view. Furthermore, the classification of a piece of information as an object or object view follows the information relativity principle (see Appendix-A), i.e. an object in one situation could be an object view in another situation. In doing so a very flexible organization structure of the product model is achieved. The general framework of a product model is of course applicable to the mechanism. The product model consists of at least a requirement model and a product structure model. A partial requirement model for the mechanism conceptual design is developed and shown. A user-interface system based upon this model is briefly introduced to show how the model is presented to and used by the end-user. (In the next chapter a mechanism structure model will be presented.) The product development model should include also a product design process model. A general design process model is presented, which founds an implementation to be discussed in Chapter 9. (This chapter also provides a context for the discussion in Chapter 9 on the issue of the relationship between tools and the mechanism structure model.)

Chapter 5 presents a mechanism structure model which is capturing and structuring the
information corresponding to the conceptual mechanism design stage. The arguments to
develop this model are illustrated, and the requirements for this model are set up. After a
short overview of the existing description of the conceptual structure of mechanisms, a new
description is proposed. Based on this new description the conceptual schema of the model
is constructed applying the hybrid data modelling approach concluded in Chapter 3. The
implementation of the model is illustrated. The connection of this model with some existing
assembly models for the computer-aided embodiment design is described. Finally, a
comparison of the model with some existing ones is presented.

Chapter 6 addresses the unique representation or identification of the mechanism structure
model when it is stored in the computer. While the design process is progressing, a lot of
mechanism solutions may be generated. The computer-aided identification of mechanisms is
helpful to avoid redundant solutions, if any. A new method with algorithms is described,
which enables to compare two mechanism structures stored in the mechanism model
developed in Chapter 5. The method with the algorithm also takes into account the evolution
of the mechanism structure while the design process is processing. The method can be used
to generate a unique labelling for mechanisms.

Chapter 7 describes the development of a graphical user interface system for the end-user to
specify the conceptual structure of mechanisms. A general discussion on the role of editor
systems in CIMOME is presented, which renders to a novel organization of editors over the
whole CIMOME. The implementation of an editor system for the conceptual structure of
mechanisms is explained in more detail. This includes:

- requirement analysis with emphasis on user-friendliness,
- explanation of the suitability of the database approach to the editor system,
- presentation of the architecture of this editor system,
- introduction to a general purpose user interface management system (UIMS) developed in
  this study and used as a tool to develop this editor system, and
- evaluation of this editor system with a summary of its novel features.

Chapter 8 describes an implementation concerning automatic generation of a finite element
model of a mechanism from the mechanism structure model formulated in Chapter 5. There
are two purposes for this implementation:

- to demonstrate the idea that almost any view of a mechanism needed for the exploration
  of the motion behaviour of a mechanism can be automated (notice: the finite element
  model of a mechanism is a kind of view of mechanisms),
- to facilitate the communication of the mechanism structure model with various tool systems
  which have been developed at Delft University of Technology and are based on the finite
This implementation, together with the editor system described in Chapter 7, facilitates the human designer in specifying a designed mechanism without a need of detailed knowledge of the underlying theory for generating the motion equation of this mechanism. The motion equation and the motion behaviour of the mechanism to be examined by the end-user are automatically generated. A tremendous improvement of the user friendliness of these systems for the motion analysis of mechanisms is obvious.

Chapter 9 describes the implementation of integration of several design synthesis tools in CIMOME. These include the tool for the cam mechanism design, the tool for the mechanism type and dimension synthesis employing FOURIER series and the tool for the mechanism dimension synthesis via optimization. The general purpose of the discussion in this chapter is to provide some verifications of the theories, methods and implementation techniques developed in the preceding chapters and to predicate perspective of CIMOME. Some salient points in particular are listed below:

- Each design synthesis tool work on a certain view of the mechanism structure model. This view is structured into a so-called design parameter model. The relationship between a design parameter model behind a tool and the mechanism structure model reflects the relationship of the object and object-view which was proposed in Chapter 4.
- The mechanism structure model described in Chapter 5 can easily be extended for these design synthesis tools if needed. This is demonstrated in the implementation of the tool for the cam mechanism design.
- An implementation of the computer-aided design process management, theoretically discussed in Chapter 2 and Chapter 4, respectively, is demonstrated. This makes the design process management more explicit, which is a unique feature of CIMOME.
- The technique for communication among design synthesis tools and communication of design synthesis tools with the CIMOME top manager component is introduced. This includes the concept such as the project database at CIMOME top level.

Chapter 10 brings together the most important conclusions and observations of the study and suggests the work to be done in future.
Design of the CIMOME Architecture

2.1 Introduction

CIMOME is a software system, so the first step to develop it is to design its architecture. The architecture of a system includes a description of (1) its main subsystems or objects and (2) the ways they interact or exchange information. The functional requirement of the system is a source to derive system components as well as their relationships.

The environment in the present study serves a means for the tool integration, see Section 1.1. Since tools are used to support a development process of a product, a software system itself or a mechanism in the present study, the environment must manifest such a development process. This makes the modern environment for the tool integration different from those following the input-output pattern described earlier in Chapter 1 with Fig.1.1 in particular. An important requirement to CIMOME is that it should support the mechanism design process. Since each tool performs a specific design task in a design process, two extra points must be addressed in the architecture of CIMOME.

- the relationship between tools, and
- the relationship between a tool and its environment.

The mechanism design process is briefly illustrated in Section 2.2, where the tools for the integration in CIMOME are also outlined.

To design, a set of design principles is always needed. Design Principles for an environment architecture are guidelines and rationales used in developing the architecture, which will be discussed in Section 2.3. Applying these principles to the integration of tools for mechanism development, Section 2.4 presents the architecture of CIMOME. Section 2.5 exposes some implementation strategies which further set the requirements on the study to be described in
the subsequent chapters. Discussion and conclusions are presented in Section 2.6 and highlight the main differences of the theory for the tool integration (via the environment) proposed and experimented in CIMOME with others.

2.2 The Mechanism Design Process and Design Tools

A mechanism is a product, so the general product development theory can be equally applied to the mechanism development process. In the present study, the term *development* is limited to only design and manufacturing. *Design* is defined as a process which generates a model of a designed object in conformity with a set of requirements (design functions and design constraints); a model of the designed object is meant for the shape and structure (hereafter these two together are called *structure* for short) of the designed object. *Manufacturing* is defined as a process that transforms an object model into a real entity, e.g. generating data for a NC-machine to produce a *cam*.

![Diagram of the Mechanism Design Process](image)

*Figure 2.1. General Knowledge of Machine Design Process*

2.2.1 General knowledge of the product design process

The general product design process can be conceptualized at two dimensions. The first dimension may be called *product life stage*. A widely accepted scheme is termed the *general*
Section 2.2  The Mechanism Design Process and Design Tools

approach to design by NN (1985,1987), see Fig.2.1 (horizontal axis). The second dimension may be called design problem solving stage. A widely accepted scheme is described in Fig.2.1 (vertical axis). A comprehensive overview and discussion of this scheme can be found in (Zandi-Nia, 1992). A design state is defined as a point in this two-dimensional space. An actual design process is therefore a path connecting those relevant design states with possible loops, see Fig.2.1.

![Diagram of a design process]

**Figure 2.2. Design Process versus Design Methods Available**

2.2.2 The Design process versus domain specific knowledge
Apart from the general knowledge of the design process described above, an actual design process is also influenced by the following factors:

- More than one method available at a problem solving stage. For example, at the synthesis stage of the conceptual design of a mechanism with a given problem, the method for the cam synthesis and the method for the linkage synthesis would both yield some solutions. The path of an actual design process could then be as such described in Fig.2.2.
- The current state-of-art of domain specific knowledge available. For example, if the required output motion contains an exact dwell more than once within one cycle of an input motion, the currently limited mechanism design knowledge won’t advise a designer to try out any solution from the bar linkage spectrum (a cam mechanism for this design problem would be recommended).
Computer-based design tools available.

2.2.3 Mechanism design problem and tools
The starting point to initiate a design process is the design problem which can be further characterized by the design requirement. One of the aspects of the design requirement is the required behaviour function that a mechanism must fulfil. In the present study, a distinction is made between function and behaviour. Behaviour of a system is defined as the conduct of a system as a result of the operation imposed on its structure (Zandi-Nia, 1992). The behaviour may be measured both qualitatively and quantitatively. For example, the customer appeal on the appearance of a consumer product reflects the behaviour of a product, which is difficult in quantification (NN, 1987). In the mechanism design, most types of behaviour can be quantified, e.g. the relationship between input and output motions. The present study is concentrating on such kind of behaviour. Function of a system is defined as the purpose of a system for its behaviour (Kuipers, 1984). In an example in distinguishing the meaning of the terms behaviour and function, Kuipers (1984) explains that the function (purpose) of a safety valve in a boiler is to prevent explosion, while its behaviour is the level which is kept at a certain height. So a function is based on behaviour and rather subjective. The explicit representation of the function of a system is not considered in the present study. The function of a mechanism which is extracted from its behaviour is then left in the human designer's mind. Hereafter, the term required behaviour of a mechanism will be used instead of the term required function¹.

In the following we shall only illustrate two classes of mechanisms design problems (more discussions can be found in Chapter 4). The purpose here is to give an impression of the complexity of mechanism design process and to introduce those tools developed or being developed at our laboratory and considered for the integration in CIMOME.

Design problem type (I).

Required behaviours:

1. Input motion type: full rotation (termed R).
2. Output motion type: reciprocating motion (termed S), oscillatory motion (termed T), or full rotation (R).
3. The input/output behaviour function, which is described in either discrete points or mathematic formula, which is termed continuous representation (Fig.2.3a,b).
4. A desired performance, which is measured from the difference between the required

¹ The function discussed so far is different from the function in the mathematic term. When the behaviour of a mechanism can be represented as a mathematic function, we shall use the term behaviour function.
behaviour and the actual behaviour.

**Required constraints:**

1. The location of the input/output bodies, the reference to measure the displacement in the machine context and the motion direction of input/output bodies. When the motion form of a body is rotation, its location means a node fixed on the ground, and its reference means a line fixed on the ground and determined with a vector (Fig.2.3c). When the motion form of a body is translation, its location means a line fixed on the ground and determined with a vector, and its reference means a node on that line (Fig.2.3d).

2. Working space of a mechanism.

3. Some other constraints such as dynamic operation constraints, cost constraints, reliability constraints and maintenance constraints. These constraints may be evaluated in a relative basis in the sense that among a set of alternative solutions the best one is chosen with respect to these constraints.

The above is called in the mechanism design theory *function generator* problem.

![Diagram](image)

Legend: → global coordinate system fixed on the machine, • fixed point, → required positive motion direction.

**Figure 2.3. Requirements of Mechanism Design Problems**

**Outline for a possible design process for problem type (I).**

The search for mechanism solution can be performed either from the various types of bar or bar-gear compound mechanisms (geared linkage) or from the cam mechanism. The process of searching for solution from bar or geared linkage can be automated and has been described by Rankers et al. (1985). A tool called TADSOL (Type And Dimensional Synthesis Of Linkage) performs this design process. The solutions generated via TADSOL must further be adapted to required constraint point (l). When the attempt to find any solution via TADSOL is failed, the design process is directed to finding solutions from mechanisms. The process to find cam mechanisms is automated by another tool called TADSO
Dimensional Synthesis of Cam Mechanisms), see Van den Berg (1981). When some solutions are found after executing the above described design process, the actual behaviour of these (mechanism) solutions is analyzed and simulated. For example the actual input-output displacement, velocity and acceleration can be analyzed and simulated by a tool called RUNMEC (Nowe, 1993). The actual behaviour is evaluated in terms of the desired performance and the required constraints. This may lead to the final decision of a solution. It would also be possible that decision has to be postponed to later design stages (e.g. the embodiment design stage and/or manufacturing design stage, etc.) due to a lack of means to evaluate solution alternatives at the conceptual design stage. Computer-based tools at the embodiment design and the detailed stages might be general-purpose commercial packages such as AUTOCAD and ProEngineer. The modification of design requirements may also be possible, e.g. reducing the desired performance, etc.

**Design problem type (II).**

*Required behaviours:*

1. Output motion is a path, to be specified in discrete points \( <x_i, y_i>, i=1,2,\ldots, m \).
2. The relationship between the input motion and the path can either prescribed or not prescribed; if prescribed, we have: \( <\alpha_i, x_i, y_i>, i=1,2,\ldots,m \).
3. The input motion type: R,S or T.
4. The same as the design problem (I).

*Required constraints:* The same as these of design problem type (I) except for required constraint point (I).

The above is called in the mechanism design theory path generator problem.

**Outline for a possible design process for problem type (II).**

First, a type of mechanism must be offered by the end-user (the designer) or by a computer-based tool, e.g. a four bar or six bar mechanism. Second, dependent on the number of points and the type of required behaviour point (2), finding the dimensions of a given mechanism may follow different processes. Optimization techniques are the most general one. A tool called OPLAM (Klein Breter & Udo, 1989b) enables to automate this design process. Next, the design process follows that described for the design problem type(I).

**2.3 Design Principles for the Architecture of CIMOME**

To design a system we are often confronted with the questions: what is a better design? and how can a better design be reached? Design principles enable to answer these questions. Nevertheless, the applicable scope of design principles is usually limited to the characteristics
of main objects of a designed system (in our case tool). The need to exhibit characteristics of these tools to be integrated in CIMOME is then obvious.

2.3.1 General characteristics of the tools to be integrated

The characteristics of the tools introduced earlier in section 2.2.3, together with future tools considered for integration, are summarized as below:

- Tools may be subject to further modifications and extensions on their functionality. For example, RUNMEC package has continuously been growing since its origin PLANAR package was developed in about 1977 (Van der Werff, 1977).
- Each tool may handle quite different data structures resulting from different application semantics as well as possible different syntaxes. For example, the TADSOL package mainly handles the Fourier representation of a behaviour function and the kinematic parameters within a standard reference system, while the RUNMEC package mainly handles a specific finite element model of mechanisms (more discussions can be found in Chapter 8).
- Some tools need intensive computation. For example, the tools based on the finite element method to simulate the dynamic behaviour of a mechanism take a lot of the computing time, while other tools need less computation but manipulate very complex data structures, e.g. a tool called EDMECH (EDiting MECHANism structure) (Zhang & Van Dijk, 1994b).
- Tools needed to support a complete mechanism design process are usually not developed by only one company or organization. For example in the scope of CIMOME we also consider integrating those tools that perform the tasks such as 2-D and 3-D drawing of parts and/or assemblies and solid modelling of geometric entities; these tools are, e.g. the well-known AUTOCAD.
- Existing tools are mostly packaged; they have their own well-defined input and output data format. Unpacking them for adaption might be inefficient or impossible.

2.3.2 A better design of the architecture of the environment

A better design of the architecture of the environment for tools integration with consideration of the tool characteristics mentioned earlier should be in general such that in the environment user's view tools are seamlessly collected to facilitate the product development, while in the environment builder's view the system has the following features:

- **Flexibility.** Modification of one tool has none or minimal influence on other tools and the environment. At the program level this implies that the system should be of high modularity.
- **Openness.** Replacement of one tool by another costs little. This also implies that inclusion of new tools costs little.
- **Concurrent development.** the total system should be divided into separate but related parts with defined interfaces and the development of the total system can possibly be based on
2.3.3 Design Principles (termed DP hereafter for short)

**DP(1). Integration dimension.** As mentioned earlier in Chapter 1 the starting point to integrate tools must be to investigate potential sharing among tools. The integration effort is no more than to impose and facilitate those varieties of sharing. The *integration dimension* is defined as the basic aspect of sharing, along which various integration schemes can be decomposed and evaluated. In the viewpoint of the tool integration, three basic aspects of sharing are recognized below.

- Tools produce and consume data about a designed object. A data repository is needed to maintain these data. Tools therefore share this common repository. To facilitate data sharing, a *data manager* should be available in the environment. This leads to the so-called *data integration dimension* (Fig.2.4). The *common database approach* used in constructing the CASE environment (Buxton & Druffel, 1984) focused on this dimension. For a tool say A, the main concerns on the data integration dimension are:
  - the data that tool A operates on,
  - parts of data which should be made persistent,
  - the format of these data,
  - other tools which also operate on these data, which implies that tool A and these other tools share common views on the concerned data, and
  - techniques to make data consistent.

A more discussion on the data integration dimension will be given later in this section.
Tools perform specific design tasks and they together formulate a (part of) the design process. Under this context tools share the design process. To facilitate this sharing, a process manager should be available in the environment. The process manager controls a design process in terms of a predefined task or action execution scenario which is represented in the computer as the process model. To understand the role of the process manager in the environment, an analogy would be helpful as shown below.

<table>
<thead>
<tr>
<th>process manager</th>
<th>car driver</th>
</tr>
</thead>
<tbody>
<tr>
<td>design scenario (process model)</td>
<td>predefined route</td>
</tr>
<tr>
<td>the current design state</td>
<td>the current site located on map</td>
</tr>
<tr>
<td>product model schema</td>
<td>car</td>
</tr>
<tr>
<td>design requirement</td>
<td>starting site</td>
</tr>
<tr>
<td>instantiated product model</td>
<td>destination</td>
</tr>
</tbody>
</table>

The difference between running a car and running an environment system in the computer for a design task is, however, that the design process of an environment is co-driven by the process manager and the human designer (the end-user). The main concerns of the process integration are therefore:

- process model,
- design state and history information, and
- method to trigger individual tools.

Both tools and the environment need to communicate with the end-user (the human designer) so that the process manager, in cooperation with the end-user, can drive a design process approaching to a design solution. Having a common look-and-feel is helpful to such communication. To facilitate this, a user-interface manager should be available in the environment. This leads to the so-called user-interface integration dimension (Fig.2.4).

The emphasis of the unification (or sharing) on the above three dimensions in defining an integration scheme may be different, dependent on applications. In practice, we have to take into account factors such as cost, time and resources to determine an actual integration scheme for implementation.

DP(2). About a single tool. Each tool should have a boundary so that the interface or communication between tools can be clearly defined. Since a tool is served as a design task or action in a design process, the criterion to make the boundary of a tool is in general the role which this tool plays in a design process. A task can be recursively divided into ever smaller tasks. Therefore in general several tasks together are served by one tool. The tasks performed by one tool usually follow one (or several closely-related) physical principle, which should be taken as a condition with a high priority in defining the boundary of a single tool.
It is usually tempting to put more tasks into one tool physically. (By physically it means that in the viewpoint of the computer program, each module is defined as a program object module and linked together to form one executable program, which is called one process hereafter for short.) The advantage of doing so is that communications between tools in such a process are relatively simpler, i.e. by following the procedure/procedure interface manner. The disadvantages are, however, that

- the system flexibility is degraded as whenever a module is changed a complete tool (process) is surely influenced,
- the complexity of a process is increased,
- performing two tasks in parallel (if any) is hindered, and
- fast prototyping, i.e. coding and testing the system to perform one specific task is difficult.

It is therefore necessary to trade-off whether to put more tasks as one process physically. The importance of this principle is to provide a guideline in developing new tools and in reorganizing existing tools in accordance with a dedicated integration scheme.

**Figure 2.5. Constructive View of Tool and Environment**

**DP(3). Constructive view of tool and environment.** The structure of a single tool can be conceptualized as shown in Fig. 2.5. Fig.2.5a shows that a single tool is composed of four ingredients which are separated but joined. In this way, the replacement of any ingredient in an entire tool system may not affect each other. The system flexibility is increased. Fig.2.5a is also supportive to the three dimension integration concept as a tool needs three types of
communications which cannot cover one another. A tool can be composed of ever small tools (sub-tools) as shown in Fig. 2.5b. This happens when a design task needs a closer cooperation among tools. Usually only a new user interface needs to be written for this new tool while the core ingredient of this new tool may just reference to composing tools. This implies that a tool may logically contain (parts of) other tools. The idea here can also be applied to the environment (or sub-environment) (Fig. 2.5c). The underlying philosophy of the above idea is to follow the so-called information relativity principle (Appendix-A) which is an important factor to increase the system flexibility.

This view of tool and environment is especially important for the tools with previously mentioned characteristics and for the environment to support the product development with more abstraction levels such as the machine development. Fig. 2.5d shows that at the general machine design stages, the conceptual design and embodiment design, there are accordingly environments A and B; the environment C consists of A & B (also see the concept of horizontal and vertical integration described in Chapter 1). The significance of this view is that a large environment can be developed incrementally and concurrently. For example, in the present study, CIMOME will concentrate on the design activities at the conceptual design stage, while in future CIMOME will grow to embody all the machine design stages.

**DP(4). Classification of types of database.** The discussion in DP(3) also implies that it is not necessary to have only one database for a large environment; sub-environments could have their own database. The database at the top environment level may only contain the information of the context of each sub-database in sub-environment. Databases are also logically classified into project database and constant database (Van der Werff, 1986). The project database contains the information of a specific product while the constant database contains the information applicable to all products in the scope. Technological data (such as standard materials with their physical properties) and catalogues of physical components belong to the constant database.

![Diagram](common_view.png)

**Figure 2.6. Common View of Data among Tools**

**DP(5). Data integration re-visited.** The goal of the data integration is to ensure that all the information in the environment is managed as a consistent whole, regardless of how parts of
it are operated on and transformed (Thomas & Nejmeh, 1992). The necessary condition as implied in $DP(t)$ is that two tools must logically have a common view or interpretation on certain portions of data (Fig. 2.6). Such a common view may also be physical in the sense that the intersecting parts of two data blocks as shown in Fig. 2.6 have the same data format. In the following, some guidelines are given to help to determine the data integration scheme considered at the implementation level in the environment such as CIMOME.

![Diagram](image)

*Note: suppose the sequence of execution is: first tool A and then tool B*

**Figure 2.7. The Concept of Local Persistent Data**

**Classification of data: data integration guideline (I)**

Globally a computer program (tool) consists of data and procedures. Data can be classified into persistent data and non-persistent data in terms of their duration with the execution of the program. Non-persistent data does not survive the execution of the program that manipulates it. The persistent data may further be classified into local persistent data and global persistent data in an environment where multiple programs interact. The difference is that the local persistent data does not survive (or does not keep its interest after) the execution of the whole environment; it only survives the execution of one or several programs. The local persistent data is usually produced in the following situation. Suppose that previously two tools A and B are physically formed as one executable program (Fig. 2.7). Their communication on data fully relies on the variable (a fundamental information carrier in the computer programming language). When these two tools are physically separated, the data encoded in the variable, which previously carries the common view of tools A and B, then becomes the local persistent data.

**Culture conflict problem: data integration guideline (II)**

Usually persistent data is stored (but not necessary) in a database manipulated by DBMS (DataBase Management System) such as ORACLE while non-persistent data is stored in the variable manipulated by a specific programming language such as FORTRAN and C. The data structures represented via variables of a programming language and those in a database are usually very different and reflect different cultures. To merge these two cultures may cost a lot of time and effort caused by the complexity of the program to be developed, such as the
unmanageable program size$^2$ and the very low data access speed$^3$ (hereafter this is called culture conflict problem). The culture conflict problem may also refer to the situation that one tool handles semantically-different application data models simultaneously. The discussion so far shows a view from the present study that care must be taken of only using database for the data communication among tools, particularly for the local persistent data.

![Diagram](image)

Figure 2.8. Information Content versus Product Development Process

**Avoidance of redundant information: data integration guideline (III)**

In the machine development, for one product part say *shaft* (Fig.2.8a), when it is handled at different development stages, the information content about this product carried with documents may not be kept constant. For example, the document about the shaft flowing to a production operator only contains the dimension, tolerance and the surface quality (Fig.2.8b), while the information such as the design intent and the relationship of this shaft with other product parts is not included. Inclusion of such information may confuse the operator. This observation can be generalized as: the information content of an object changes with the design stages (Fig.2.8c). Redundant information of tool A means a part of the information not necessary for the operation of tool A but necessary for the operation of tool B. The redundant information potentially decreases the system modularity and thus flexibility. For example, when tool B is changed, this change may need to be propagated to tool A due to the redundant information (with respect to tool A) which is, however, needed for tool B. The redundant information may also reduce the system openness, because when tool A is replaced by tool A' with possible more functionalities, an extra cost needs to be paid on the syntactic adaption of tool A' for the redundant information possibly existing in tool A. It should therefore be avoided to carry redundant information into a tool.

---

$^2$ This is particularly important when the prototype system is being developed on the personal computer with MS-DOS system and much attention on the portable source code.

$^3$ Undesirable data access speed using a commercial relational DBMS was reported by Kaper (1987).
Appropriate use of programming language file: data integration guideline (IV)
When the information content is little, the corresponding syntax becomes simpler. When the data syntax can be constructed as simple as the format with a series of sequential records (hereafter file for short) and the type of data is local persistent, use of such a file instead of database would be preferred. The reasons are given below:

- The culture conflict problem does not exist, assuming that the handling of such a file is obligatory to any tool constructed with any type of programming languages.
- The system performance such as data access speed can be improved particularly when the quantity of data for communication is large.
- The adaption to the change of one tool for another is easy.
- The concurrent development process can be facilitated because on this type of file, agreement on data communication will be more easily reached among different tools.

\[
\begin{align*}
\text{(a) the external conversion} & \quad \text{(b) the internal conversion} \\
\text{Legend:} & \quad \rightarrow \text{conversion module from data A to data B} \\
& \quad \bigcirc \text{ tool; } \square \text{ data block; } \leftrightarrow \text{ tool and data interface}
\end{align*}
\]

*Note: in the case (b) the conversion module is embedded in tool B. A new tool B' consists of tool B and the conversion module.*

Making data conversion process module external: data integration guideline (V)
Utilizing both (programming) file and database as the means for data communications requires to have a data conversion process between file data to database data or vice versa. There also exists a conversion issue between different databases. The conversion process is better made external in the sense that it is an independent program module between tool A’s data and tool B’s data (Fig.2.9a). The advantage of doing that is to increase the system flexibility. For example, when tool A is changed, it is most likely that tool B should be changed when the conversion process is not made external (Fig.2.9b). The conversion process is also responsible for maintaining data consistency between two relevant blocks of data. To be more
generalized, we shall propose the term data link as an embodiment of the total meaning of
data conversion. A set of processes can be attached to the data link. A data link manager in
an environment is responsible for maintaining data links, e.g. invocation of a data link
process to maintain data consistency between two data blocks whenever needed.

DP(6). Process/Process interface method. As we mentioned earlier in DP(2), a tool may
be a program module or process. It is therefore necessary in an environment to define the
process/process interface. An interface protocol needs to be enacted and a manager in an
environment is responsible for enacting such a protocol. Although some operating systems
provide methods for process/process interface, a higher level protocol is helpful to achieve
more portability of an environment.

2.4 The CIMOME Architecture

The design of the CIMOME architecture now becomes a decision-making task by applying
the principles formulated in the last section in the context of integrating tools for design and
manufacturing of mechanisms.

2.4.1 An adapted classification of tools

Earlier in Section 2.2.3 we have introduced the existing tools which are expected to be
integrated in CIMOME. The state of these tools is that they have a powerful computation
capability but the user-interface is not mature. Therefore it provides us opportunities to make
some adaption of their definition in accordance with the architecture design principles.

In viewpoint of the functions (or roles), these tools are classified into three categories: design
synthesis, analysis and specification.

In the design synthesis, there are:

- Cam design (CAMDES). This is a modified tool from the existing TADSOCP program.
- Linkage design (LINKDES). This is a modified tool from the existing TADSOL program.
- Linkage Dimension Synthesis via optimization (DSYNOP). This is a modified tool from
the existing OPLAM program.
- Embodiment Design (EMBDES). This tool works out a preliminary layout for a planar
mechanism (Huang & Zhang, 1993) and it operates on the interface between the conceptual
and embodiment design stages.

In the design analysis, there are:

- Kinematic analysis (KINANA). This a modified tool from the existing RUNMEC program,
particularly the part corresponding kinematics. This tool can also be used for synthesis, e.g. parameter variation.

- Dynamic analysis (DYNANA). This is a modified tool from the existing RUNMEC program, particularly the part corresponding dynamics.
- Mechanism Identification (MECHID). This is a modified tool from the program comparing two mechanism structures on the topology (a more discussion can be found in Chapter 6).

In the design specification, there are:

- An editor for the specification of an artifact including mechanism structures and design intents. The design intent is the purpose of design on artifact structures. For example, specification of a shoulder on a shaft is to axially position a bearing. Chapter 7 has a detailed description of the development of this tool (EDMECH).
- An editor for the specification of requirements including required behaviours and design constraints. So far an editor (termed EDFUNC) for the specification of required behaviour functions has been developed (a more discussion can be found in Chapter 4).

According to DP(2), part of tools above are viewed as sub-environments of CIMOME, e.g. CAMDES, LINKDES, DSYNOP, KINANA, etc. From the viewpoint of the construction of tools, also see DP(3), we have sorted out a kind of service tool in CIMOME (note that the tools mentioned above are called design tool). The features of the service tool are that

- a service tool corresponds to action or activity in a design process, and
- a service tool does not need the participation of the end-user, which implies that the task performed by a service tool is completely automated.

Examples of service tool in CIMOME are: the part of RUNMEC which performs the kinematic analysis and a program called GMMFEM (details are referred to Chapter 8) which automatically generates a specific finite element view of mechanisms, i.e. an input data file for RUNMEC. The relationship between design tools and service tools is many-to-many. Service tools actually symbolize the design automation, while the existence of other tools with user-interface shows that a full design automation is not researched so far.

Some other tool modules are needed to support the execution of CIMOME. They are:

- the file management
- the browsing of a design (design object, design history, etc.), and
- the user-readable document generation.

Further remarks regarding the above classification are made below:
Section 2.4 The CIMOME Architecture

- It has been observed that the modelling of mechanisms for the kinematic analysis and
dynamic analysis based on the finite element method (hereafter FEM for short) (Van der
Werff, 1977; Klein Breteler, 1987) follows slightly different principles (details are referred
to Chapter 8). Applying DP(2), we shall define the kinematic analysis and dynamic
analysis as two separate tools. It is possible, however, that physically they share a lot.
- The design synthesis tool may contain the part which makes more sense of analysis. For
example, in LINKDES (or TADSOL), when a mechanism is worked out from a case
mechanism catalogue, the actual input/output behaviour function is produced by an analysis
part (Klein Breteler, 1983b). Although the function of this part can be overridden by
KINANA tool, it is kept with LINKDES for the efficiency purpose.
- Logically the above tools may contain each other. For example, tool CAMDES logically
contains EDMECH and EDFUNC as it is necessary to specify a type of cam mechanisms
and a required behaviour function for calculating the cam profile and other kinematic
parameters of a cam mechanism. This is in accordance with DP(3).

2.4.2 A functional view of the CIMOME architecture
Gathering the tools with the several system components, a functional view of the CIMOME
architecture is presented, see Fig.2.10. Fig.2.10a is a three-projection view; tools are plugged
into the tool slot, which implies a highly flexible and open architecture. Fig.2.10b shows that
a design tool itself can also be viewed as an environment, which is in accordance with DP(3).
The design tool has all the system managers that the environment has. The function of the
system managers of the environment is therefore a union of the functions accomplished by
the system managers at the level of design tools. Such an architecture therefore allows for a
concurrent systems development (one of features of a better architecture design described in
Section 2.3.2). In the following, the functions of the system managers (or agents) and the way
they work together, in the context of CIMOME, are elaborated.

User-interface manager
User-interface manager is responsible for (1) displaying the gadgets which are menu, slide
bar menu and text entry box, etc. to show the current design state to an end-user and wait for
the end-user’s action, (2) interpreting the end-user’s intent via these gadgets to enact a user-
initiated design process and (3) driving a predefined dialogue sequence. The user-interface
manager works on a user-interface model which contains the scenario of the user-computer
interaction. More details about the user-interface manager as well as the user-interface model
are referred to Chapter 7.

Process manager
Process manager is responsible for assisting the end-user to execute the design process, as
described in DP(1). The process manager works on (1) a process model which contains the
scenario of the design process for a specific design problem and (2) a record of the current
design status. So the work of the process manager can be further decomposed, see Fig.2.10d,
where the process/process protocol executing agent will be explained later in this section. The process manager may report to the end-user if the process manager finds incorrect direction of a design process initiated by the end-user, based on the scenario in a process model and the record of the current design status; the process manager triggers individual tools to progress the design process. In CIMOME, the process manager is served as a main program in the viewpoint of the computer program organization. More details about the process model are referred to Chapter 4 and an implementation can be found in Chapter 9.
Process/process interface protocol and execution
In terms of DP(1) an DP(2), the CIMOME program system won't be as such, all the tools are defined as program object modules and linked to be one executable program. In CIMOME, each tool itself is a program process and the environment is of course a process too. The process/process interface protocol must be defined in CIMOME and is as follows:

- The communication between two processes must be effected via the protocol execution agent who is like a central-controlled telephone operator. This implies that two parties engaged in communication do not directly contact.
- Messages from one process to another must be put into a file which is like a mail-box. The definition of this file includes the following information:

  - sender process name,
  - receiver process name,
  - purpose (specifying what is required to do from the sender),
  - response (feeding back the information about the performance of a receiver), and
  - a list of files (including: file name, location, file semantics and file syntax) a receiver process needs for its operation.

- The current receiver process is responsible for erasing the message sent by the sender process to avoid the message overflow in the mail-box.

To facilitate the control of a design process, the process/process protocol further imposes:

- all design tools can communicate with the process manager,
- the communication between design tools (except for EDMECH and EDFUNC) must be effected via the environment process manager, and
- design tools can communicate with service tools directly.

The reasons to allow for a direct communication between one design tool with tools of EDMECH and EDFUNC are:

- they are generic in the sense that the information provided by them is a common need to all other design tools, and
- a design tool can be viewed as an environment. In this sense, the design tool should be allowed to communicate with the tools that (i.e. EDMECH and EDFUNC) are common to all design tools.

However, after a design tool has been directly communicating with EDMECH and EDFUNC, results (e.g. new design solutions or any change on requirements must be informed of the process manager at the environment top level by updating the information stored in the
database at the environment top level. In other words, at the level of each design tool (notice that it can also be an environment) there may exist a local database corresponding to this design tool. Databases at the environment top level are called global (or central) database. By following the *information relativity principle* (Appendix-A) the distinction of local and global is of relativity, which implies that today’s global database may become tomorrow’s local database with the growth of an environment. As a result, a hierarchical organization of databases is reached (Fig. 2.11). So far in CIMOME, there only exist two levels.

![Diagram of hierarchical organization of databases in CIMOME](image)

*Figure 2.11. Hierarchical Organization of Databases in CIMOME*

**Data manager versus data link manager**

As indicated in Fig. 2.10e, the data manager consists of three parts, of which the data link manager needs further clarification. *Data link manager* is responsible for triggering appropriate processes to maintain data consistency. It is functionally supplementary to the data manager. The data link manager differs from the service tool mainly in that the data link manager performs data conversion with respect to different representations of the data with similar meaning. Two blocks of data which need to be maintained semantically consistent could be:

- two databases,
- one database and one file,
- two (programming language) files

In terms of the above explanation, tools like RUNMEC and GMMFEM do not belong to the category of data link processes. An example of the data link process can be identified in
CIMOME. In the dimension synthesis via optimization performed by tool DYSNOP (Wagensveld, 1993). The application data model underlying DYSNOP describes a set of design variables which are to be optimized (Fig.2.12 on the right side). Such a model is called design parameter model (a more discussion can be found in Chapter 9). Two processes ($DL$ and $DL'$) are set between the mechanism structure model (see Chapter 5) and this design parameter model to maintain the semantic consistency, i.e. when any change takes place in tool DYSNOP, corresponding data in the mechanism structure model must be updated and vice versa. These two processes belong to the category of the data link process. In CIMOME, the determination of whether physically to set a data link process external or internal follows the strategy in the following. When two tools, handling two data blocks related to a data link process, are independent with respect to these two data blocks, the data link process is preferred to be set to be external, physically.

Help agent
Help agent performs some generic tasks (Fig.2.10c) necessary to execute CIMOME. It differs from the process manager in that the help agent provides helps for the end-user to communicate with the process manager, e.g. the end-user can know the current design status via browsing facility. The document generator produces user-readable design documents like a technical report which essentially contains (1) design requirements (2) design solutions (if any) with design intent and (3) design rationale with design history.

The way that all managers work together
Help agent helps an end-user to register in CIMOME with user-id, date/time, etc. A design process is initiated by the end-user. The user’s action is, through the user-interface manager, sent to the process manager. When the process manager agrees with the direction of the end-user, it triggers an appropriate tool program via the protocol execution agent. At the level of an individual tool, such a procedure may happen again since the tool itself would be an
environment; a design process is then extended recursively in depth. A tool needs resources (pre-conditions) for operation and produces new resources (post-conditions). Those resources are data about a designed product (a mechanism in the present study) which are managed via the data manager with the help of the data link manager.

2.5 Implementation Considerations

The main demand in consideration of the implementation of CIMOME is to have a fast prototype which should not only be practically accessible to most small and medium companies but also provide some verifications of the idea described. Taking this in mind, it has been decided for the time being not to rely on those commercial systems such as windowing systems, commercial DBMSs and UIMSs because of

- unjustifiable commercial availability,
- unsatisfactory data access speed,
- portability, and
- ease of compatibility with our exiting tools which were developed mostly in the FORTRAN language in the past twenty years.

Our own developed systems corresponding to those commercial systems are used, which are listed below:

- A FORTRAN-based Graphic Mini-System (GMS). This was developed with the objective to achieve a highly portable system (Klein Breteler, 1993).
- A FORTRAN-based database management system (hereafter this is called ABC for short and in Chapter 3 there is a short description).
- A FORTRAN-based User-Interface Management System (UIMS) (in Chapter 7, there is a short description).

For the system development we follow the so-called data-centre (or generalized object orientation) approach, in which we shall first formulate a mechanism structure model (Chapter 5) and make a user-interface upon it (Chapter 7). All other tools are then in parallel brought into the environment in a unified way, namely (details can be found in Chapter 9):

- to define an application database model for the application semantics that those tools are handling, e.g. the design parameter model,
- to modify those tools in accordance with this application database model, e.g. the communication with the mechanism structure model and the part of user-interface, etc.

Application database models that tools operate on reflect aspects or views of a designed
object, mechanism in particular. These application database models, together with the mechanism structure model (Chapter 5) form a so-called mechanism product model (Chapter 4) similar to the product model in the general CAD/CAM field.

2.6 Discussion and Conclusions

The concept of the environment for tools integration has been of interest in CASE since the beginning of the 1980s. In the mechanical engineering domain, a nearly similar issue has been studied at the same time, that is, developing computer-based systems for integration of CAD/CAM. CIM could be viewed as an extension of the present study by considering more integrated elements, e.g. the management and production activities and so on. More recently in the mechanical engineering domain, a more ambitious concept virtual manufacturing is introduced by Kimura (1993) and Onosato & Iwata (1993). The nature of this concept is much more to utilize (than ever before) the computer and information technology in the simulation of various activities of product development. In this context and without limiting to the mechanism development, several features of the CIMOME architecture with its underlying design principles can be identified and summarized below with discussion of some related work and some interesting observations.

- The integration context of CIMOME is different from all other work (Tomiyama & Ten Hagen, 1987a; Onosato & Iwata, 1993; Kimura, 1991, 1993; Oh et al., 1994; Marechal, 1987; Cutkosky and Tenenbaum, 1991; Ehrlenspiel & Schaal, 1991; Wei et al., 1990). The CIMOME architecture with its underlying design principles considers both the horizontal and vertical integration (see Fig.1.3), while most of other work is mainly focused on the vertical integration. In particular, we start with the implementation of integration of tools at the conceptual design stage which is widely recognized important and crucial in the whole cycle of product development (Tomiyama & Ten Hagen, 1987a; Oh et al., 1994), while most of other work start with the embodiment design stage. The element to be integrated in CIMOME is the tool. This is what we have learned from the software engineering and is to make explicit in the mechanical engineering domain.

- In the CIMOME architecture, a design process is defined with consideration of not only the general design stage (most other work is concerned with) but also the things like tools available and domain-dependent knowledge; a similar observation is recently made by Blessing (1993). As a result, the content of our process model will include domain-specific knowledge. Since the process manager operates on the process model, CIMOME supports the design process management for domain-specific problems, e.g. the mechanism design problems mentioned in Section 2.2.3. Although in a proposal for III-CAD (Tomiyama and Ten Hagen, 1987a) a system component design supervisor with design scenario is much similar to the process manager with the process model in CIMOME, they did not explicitly relate the design scenario with tools. In the architecture of III-CAD, tools (in their term
applications) are passive in the sense they are only invocable by the designer supervisor and corresponding to the service tool in CIMOME. (This is further related to the discussion of the next feature of the CIMOME architecture later.) In our view, support to the design process along the general design stage is not much helpful to the human designer, because such knowledge is quite well-known to the human designer. Instead, a much useful support to a CAD system should manage a design process towards solving domain-specific problems. Moreover, for the computer to support the human designer along the general design stage, effort must be put on the data integration dimension or the maintenance of the data consistency. This might be the reason that today many CAD/CAM systems only consider a so-called product model, e.g. Lindberg and Agerman (1992) stated that the product structure is the backbone of CIM.

- It is a novel idea to view tool and environment by following the information relativity principle (Appendix-A). This infers that a tool can be an environment and therefore can actively drive, with the end-user, a design process. It is quite important to the context of CIMOME that the implementation of integrating those tools can be concurrently carried out. When this view is applied to the environment itself, a philosophical foundation to develop a larger environment incrementally has been established.

- It is novel idea to clarify the process manager by making an analogy with driving a car. Many different arguments can be found in the literature relevant with the product development process. Kimura (1993) implies that the content of process modelling is such as the modelling of metal cutting process and the finite element modelling. He further distinguished the process modelling from the activity modelling. The activity modelling is concerned with the modelling of the human activity in a product development process, e.g. the process model by Mi & Scacchi (1992). In our view, The difference between the process modelling and the activity modelling lies in different views in different contexts. At the time the computer was not well developed the finite element modelling could be a manual task performed by the human designer and thus a design activity. So an activity in the past might be the process nowadays. Therefore in CIMOME we only use the term process which also includes the activity.

- There is heated argument about whether a CAD system must be process-oriented or product-oriented. In the past the product model concept in integration of CAD/CAM was appreciated (Kimura et al., 1984; Lindberg & Agerman, 1992). The recent literature in favour of the process-orientation are (Blessing, 1993; Geddam & Kaldor, 1993). Unfortunately in the current literature, a clear definition is given neither to product-orientation nor to process-orientation. Usually the product-orientation is concentrated on the modelling of the product data while the process-orientation is concentrated on the modelling of the design process. The so-called process-orientation by Geddam & Kaldor (1993) actually means that development of a product must be restricted with limited and available manufacturing processes, which seems a bit away from the issue here. In developing the CIMOME architecture we have considered both: product and process. This is reflected in our idea called three dimension of tool integration (Fig.2.4). Distinction
should be noticed between the way a product is developed in reality and the way of developing a CAD system to support product development. In the point of view of the former a product is the result of executing a design process. For the latter we shall refer to two well-known methodologies function-orientation and object-orientation. In implementing CIMOME we follow the object-orientation as mentioned in Section 2.5. We have found that the fact project management puts much more weight on the success (Cooper, 1990) is not sufficient to require the function-oriented system development approach. The object-orientation system development approach does not exclude the description of process as well as the control of processes, while it puts priority as:

\[ \text{data} \rightarrow \text{process} \rightarrow \text{control} \quad \text{(Winblad et al., 1990).} \]

Such priority comes up with many advantages; among them, remarkable is the one supportive to the concurrent system development.

- In the point of view of the system architecture for an integrated system, the history of developing the computer-support system follows:

\[ \text{input/output pattern} \rightarrow \text{product-orientation pattern} \rightarrow \text{product with process pattern} \]

In the first pattern the data is implicitly related to a design product and the process is sequential. In the second pattern the data is strongly related to a design product but the process is not explicitly represented in the computer. As a result the system is merely a big pool in which computer-based tools stay and are invokeable by the end-user; the design process is completely in the mind of the end-user. In the third pattern (i.e. the one CIMOME belongs to) both product and process is co-driven by the end-user and the computer. The result of execution of a design process is an instantiated product model.
Data Modelling: an approach to CIMOME

3.1 Introduction

This chapter provides an approach to the CIMOME realization. This approach is termed database approach and it will be widely employed in later chapters for the data representation of various machine objects. Simply speaking the approach is to extensively exploit the database design methodology in the system development. This is very much geared to the so-called data-centred system development since the mid of the 1980s. The idea also gives some implications to the future computer system development, that is, less source code but more data modelling. In Section 3.2 basic concepts regarding database are clarified. Importance of conceptual data modelling is recognized. In Section 3.3 a set of building blocks for the conceptual data modelling with their underlying concepts is introduced. Since it has not yet been found that a known data modelling system covers this set, a term hybrid is referred to the approach based on this set. Some guidelines for the database design are also included in this section. In Section 3.4 a particular need of database modelling to CIMOME realization is illustrated. The section ends with a conclusion that a tool for the main memory data management is necessary. The development of this tool is briefly presented in Section 3.5. Section 3.6 gives discussion and conclusions.

3.2 Definition and Discussion of Basic Concepts of Database

Definition 3.1. Database (informal). A database is a collection of related data. Its function is to represent the interesting semantics of an application, the mini-world, as completely and accurately as possible (adapted from Dittrich, 1986; Ter Bekke, 1992a).
Conventionally the main activity of a system developer is to design data structures known to the programming discipline to represent various application objects and methods. The function of a database implies that the database developing method can be a useful tool for the system development. In the aspect of data manipulation, the database system actually provides some unified means with generic operations: creation, deletion and updating. This leads to a philosophical basis that in developing CIMOME the database design replaces the data structure design as a major system development activity. (In the present study, the term data representation may be used as a synonym of database.)

Definition 3.2. Data Model. A data model defines a framework of concepts to be used to express the mini-world semantics (Dittrich, 1986). In the present study, those concepts are limited to a collection of data structure types. An application database is established using a data model, such as the relational model (Codd, 1970) and the semantic data model (Hammer & McLeod, 1981; Ter Bekke, 1992a).

Definition 3.3. Data Modelling. Data modelling is a procedure to use a data model to define an application database. The data modelling method contains a set of schema building blocks with guidelines to use them to define an application database. Data modelling is in this sense database design or developing data representation of application semantics.

**Figure 3.1. Type, Class and Instance**

Definition 3.4. Type, Class, Attribute and Domain of Attribute. Types denote a structure and a domain of instances of that structure, while classes are a named collection of objects of a type. It is possible to have more than one class of a type (Liu, 1993). Fig.3.1 shows the relationship between type, class and instance. Base type constructors are, e.g. Integer, Real and String, etc. The definition of a type (non-base) includes type name and a list of function descriptions each with this type name as the leading input argument (Liu, 1993). A function of a type is also termed as attribute of a type. In some semantic data models (Hull & King, 1987) attribute is also called role. Attributes may further be classified into member attributes and class attributes. A Member attribute describes an aspect of each member of a class, while a class attribute describes a property of a class taken as a whole (Hammer & McLeod, 1981). The definition of a class is straightforward by relating a class name to a type name and in most cases these two share the same name (in particular for non-base type), e.g. in the syntax by Liu (1993) class PERSONS: {PERSON}. For this reason a strict separation of type and
class is not emphasized in the modelling activity of the present study. Domain of an attribute is all the values that this attribute can take.

**Definition 3.5. Database (formal)** (Liu, 1993). A database consists of two parts: database schema and database instance. A database schema is described by a list of type and class descriptions. A database instance consists of a set of classes each associating with a schema type in the database schema.

**Definition 3.6. Database design phase.** Database design is usually divided into three phases, i.e. designing conceptual schema, external schema and physical schema. Ter Bekke (1992) made clear for each phase, their purposes and main concerns (Fig.3.2).

![Diagram showing the Three Schema Database Architecture](image)

*Figure 3.2. The Three Schema Database Architecture*

Such a division is a natural evolution of the database design methodology, as database is being utilized in more complex applications, such as the machine design in the context of the present study. A comparison between the database design and the machine design is made by Zhang & Van der Werff (1994c), as shown below.

<table>
<thead>
<tr>
<th>machine design</th>
<th>database design</th>
</tr>
</thead>
<tbody>
<tr>
<td>conceptual design (solution principle, etc.)</td>
<td>conceptual design (conceptual schema of application semantics)</td>
</tr>
<tr>
<td>embodiment design (layout, etc.)</td>
<td>logical design</td>
</tr>
<tr>
<td>manufacture design (process planing, etc.)</td>
<td>internal design</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>internal schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>defining data implementation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>conceptual schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>defining data relationship (i.e. semantics)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>external schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>defining data interpretation</td>
</tr>
</tbody>
</table>

The above discussion has sufficiently implied the importance of the conceptual database schema development, which will be focused on in the present study.
Definition 3.7. Conceptual data modelling. It is a process to develop a conceptual database schema using data models. Not all data models are suitable for the conceptual data modelling. For example the relational data model was found not suitable for the conceptual data modelling (Hammer & McLeod, 1981; Ter Bekke, 1992). Semantic data models are suitable for the conceptual data modelling as they are able to capture more application semantics.

Definition 3.8. Database designer and database user. A database designer is a person who defines a database schema. A database user is a person who actually creates instances in conformity with a defined database schema. A database user may not be aware of an underlying database schema or he even does not know that he works in the database environment. This implies that usually an interface is built upon the database schema. The database user is a machine designer or system developer who makes computer programs to communicate with a database schema, dependent on a specific application context.

3.3 A Hybrid Approach to Developing Conceptual Database Schema

Several observations are described to show that each data model misses something important to the current application. This implies the need of a hybrid approach to develop CIMOME. Next, a set of modelling notions necessary to the present study is described. Finally some guidelines are given regarding the use of them.

3.3.1 The need of hybrid

Observation-1
A set-valued attribute is the attribute which can have a set of values (instead of only one) as a property of an entity. The set-valued attribute notion has been supported by some relational DBMSs and also supported in EXPRESS, but its significance to conceptual data modelling with respect to the object relativity principle (Smith & Smith, 1978) seems not well illustrated. In other words the set-valued attribute is introduced in some semantic data models\(^1\) as only a kind of abstract data type (e.g. in EXPRESS). Zhang & Van der Werff (1994c) found that lack of the support for the set-valued attribute notion at the conceptual modelling level may lead to a limited guideline of the object relativity principle, for details see Section 3.3.5.

Observation-2
Data abstraction grouping has neither been supported in the Object-Oriented data model (OO

\(^1\) The survey by Peckham & Maryanski (1988) has shown that most of semantic data models do not support rich constructors for unstructured object (or atomic object) representation, e.g. set-valued attribute.
for short) nor in EXPRESS, although it was supported in the Semantic Data Model (SDM for short) (Hammer & McLeod, 1981) and an extended Entity-Relational model (Storey, 1991). The grouping data abstraction was found important to define composite objects, see Zhang & Van der Werff (1994c), for details see Section 3.3.5.

Observation-3
The capability that data models support the composite object representation is very important in computer-aided engineering design (Barsalou & Wiederhold, 1990; Lockemann et al., 1985; Zhang & Van der Werff, 1994c). However, EXPRESS does not explicitly provide the mechanism to define composite objects, see Chan et al. (1994). The object-oriented data model does not provide sufficient building blocks for defining composite objects to capture the more dynamic nature of a lot of applications (Ter Bekke, 1994). New studies are needed, see Section 3.3.4.

Observation-4
There are many confusions in the terminology used in the data modelling. To name but a few, in (Liu, 1993) aggregation reference between two objects actually means part-of reference in (Kim, 1991); association reference between two objects means aggregation in (Smith & Smith, 1977), while association is used by Brodie (1984) and Storey (1991) for the grouping data abstraction.

Observation-5
There are also some confusions in data modelling concepts. For example, Storey (1991) distinguished generalisation from sub-type, while most of the researchers (Smith & Smith, 1977; Hammer & McLeod, 1981) did not, and still others (Hawryszkiewycz, 1984; Shlaer, 1988) were in favour of adding extra attributes to make two sub-classes disjoint (if any).

3.3.2 Conventional building blocks in the hybrid approach
Building blocks are notions with representation means (diagram and/or text) for expressing application semantics. They are here considered to be divided into two sets: conventional and non-conventional. The first set includes the notions such as:

- class, attribute, etc (see Definition 3.4).
- member attribute (see Definition 3.4 and (Hammer & McLeod, 1981)).
- class attribute (see Definition 3.4 and (Hammer & McLeod, 1981)).
- set-valued attribute (see the discussion of observation-1 above)
- domain (see Definition 3.4 and (Date, 1981)).
- generalization or is-a link (Smith & Smith, 1977; Peckham & Maryanski, 1988).
- aggregation or has-a link (Smith & Smith, 1977; Peckham & Maryanski, 1988).

The representation diagrams for these notions are brought together in Fig.3.3. Several
Figure 3.3. Diagram Representation of Basic Building Blocks

Figure 3.4. Comparison of Diagram Representation of Type

remarks concerning this figure are made below.

- In the representation diagram, type may be explicitly expressed only when the names of a class and its underlying type are different. In the database modelling there exists an issue how to diagrammatically represent two attributes (of a class) with the same type. For the example with the semantics: (1) 'managing' consists of 'manager' and 'managed-staff' and (2) both 'manager' and 'managed-staff' are the 'employee' of a company. Fig.3.4 shows the approach used in the present study (Fig.3.4a) and the other by Hull & King (1987) (Fig.3.4b). The key point of approach (b) is that a type is always explicitly expressed while attributes are represented as a label of the edge connected to two corresponding types.

- In the representation diagrams, some other types of class (e.g. 'select' type) are not shown, which are found from EXPRESS and may be useful in the present study².

- In the representation diagram, an upper-case string in thick rectangle box indicates that the

---

² EXPRESS provides a rich set of building blocks. The hybrid approach here might be thought of as an extension and variation of some building blocks of EXPRESS, so a detailed description of these building blocks is referred to (ISO-TC184/SC4/WG5, 1991).
class name is the same as its underlying type name, while a lower-case string in the thick rectangle box indicates only a class.

The second set of building blocks includes the grouping data abstraction and the composite object which are less discussed in the literature but important to the present study (see also observation-2 and observation-3 above).

3.3.3 Grouping data abstraction
Grouping is also called association by Brodie (1984). For simplification of the following discussion, an application case is described first and will be used to clarify the concept.

An application case description: Committee-People.

(1) There are several people and several types of committee (A,B,C),
(2) Each person among this set of people participates in only one committee.
(3) Each committee consists of several people with one chairman.

A class with type 'PEOPLE' is defined first (Fig.3.5a).

![Diagram](image)

*Figure 3.5. A Comparison of Several Data Representations for Grouping Class*

Hammer & McLeod (1981) described that a grouping class (G) is defined as collecting the members or instances of an underlying class (U) based on having a common value for one or more designated member attributes of U. A grouping expression specifies how the members of U are to be placed into these groups. For the above case a class type called 'COMMITTEE' can be defined by taking it as a grouping object and the underlying object as 'people', with the grouping expression being common value of 'committee-name' (Fig.3.5b). The model shown in Fig.3.5b misses the information about the committee name in the grouping class object 'PEOPLE'. It was therefore advised to list the name of the groups in the specification of a grouping object such as (in terms of their textual syntax):

```
COMMITTEE
  interclass connection: grouping of PEOPLE
```
on common value of 'committee-name'
groups defined as classes A,B,C
member attributes
  average-age : INTEGER

Because specific types of committee are not appeared in the member of the grouping class object, it is therefore impossible to know what a specific type of committee is by simply looking at a member of the grouping class object 'COMMITTEE'.

Hull & King (1987) defined a model as shown in Fig.3.5c (their original syntax has been converted into the one used in the present study). The key point in their definition is that the information about the committee name (in Hammer & McLeod (1981), group name) is put into the instance of the class object 'COMMITTEE'. This proposal appears that the grouping object 'COMMITTEE' is formulated with two steps. The first is to get groups of people who sit in a certain committee. The second is to define these groups as somewhat a set-valued attribute of 'COMMITTEE' (notice that there is no direct connection between classes 'COMMITTEE' and 'PEOPLE' in general). This proposal is slightly different from that by Hammer & McLeod (1981). Brodie (1984) defined that association (or grouping) is a form of abstraction in which a relationship between member objects is considered a higher level set object. Peckham & Maryanski (1988) indicated that criteria needed to decide the set membership could be like 'committee-name' (of 'PEOPLE', Fig.3.5a) = 'A' or 'B' or 'C'. (This is different from the grouping expression by Hammer & McLeod (1981).) In doing so, a higher level grouping object such as 'COMMITTEE-A' or 'COMMITTEE-B' can be defined. The set membership of a grouping object can also be controlled by the database user. This implies that the grouping data abstraction is quite user-oriented (in Section 3.3.4 this feature will be exploited in the specification for complex object).

The above discussion has shown that there is no unified building block in defining a grouping object as well as a relationship with its underlying object, although the concept of grouping is agreeable, that is, an abstraction of the members of its underlying class. In the following a description of grouping is introduced for use in the present study. The major elements of the grouping definition are (see Fig.3.6):

1. One or more underlying classes $U_i$ ($i=1,2,\ldots,n$), $n=$number of underlying classes. For example, in a previous case 'Committee-People', 'People' belongs to the underlying class.

2. Links connecting a grouping class $G$ to $U_i$ ($i=1,2,\ldots,n$). These links are abbreviated as member-of link.

3. Grouping expressions are applied on each link, which determine the set membership of instances of $U_i$, for the grouping class.

4. Other attributes can be defined with the grouping class, such as 'number-of-committee-
member' (in the application case discussed above). This kind of attributes applies to a set (of instances), instead of each individual instance in a set. Fig.3.6 shows the standard pattern of a grouping instance (in the right side of this figure).

![Diagram showing the definition of Grouping]

**Figure 3.6. Definition of Grouping**

Some remarks regarding the grouping expression and the group names are as follows:

- The *form of a grouping expression* is of broad sense, and it can be:
  - a phrase such as 'common value of TYPE' (where TYPE must be an attribute name in the underlying class),
  - a predicate such as 'committee-name'='student-adviser' (where 'committee-name' is an attribute name in the underlying class), and
  - complex statements which may involve some classes beyond the underlying classes currently considered.

- The grouping expressions are specified by taking two options:
  1. specified at the database schema definition level by a database designer, and
  2. resided in the database user's mind; this implies that in a database schema for the grouping definition, no grouping expression is specified.

For option (i), in the case of more than one underlying class, grouping expressions must be compatible to each other. The compatibility of grouping expressions gives rise to a complex situation in the data management. In the present study, it is required that a common predicate be used. For option (ii), it is better for an interactive data management system to record the grouping expression given by the database user, with an auxiliary attribute. Creation of such an attribute would be transparent to the database schema designer and therefore be the responsibility of DBMSs.

- Group names corresponding to groups of instances that are contents of the underlying classes should be made explicit in a database either at the schema definition level or at the
database instantiation level. There are several situations which might occur.

- A group name is taken from the common value which is explicitly specified in the grouping expression, e.g. for the grouping expression of committee-name = 'A'. Value 'A' will be a group name.
- A group name is taken from the common value which is implicitly specified in the grouping expression, e.g. for the grouping expression of common value of 'committee-name', actual values of 'committee-name' might be unknown to the database designer but depends on the instantiation of underlying classes 'PEOPLE'.
- In the case that grouping expressions are not given at the database schema definition level, group names are solely the database user's responsibility.

- An auxiliary attribute to record group names should be the matter at the logical database design level where a specific DBMS is chosen. This implies that a DBMS may implicitly create an auxiliary attribute to record group names.

![Diagram](image)

**Figure 3.7. Examples of Group Names as Attribute Values**

- Group names, as attribute values in a grouping class, may not be unique, which differs from the underlying reason for the schema (Fig.3.5c) by Hall & King (1987). Fig.3.7 exhibits an application case in the machine design. In Fig.3.7a, it has been shown that a mechanism can be decomposed into three groups (I, II, III). Both group I (body-1 and body-2) & II (body-3 and body-4) belong to a group type A and group III (body-5 and body-6) to group type B. (Such a decomposition follows a fundamental theory in the mechanism design and founds a lot of mechanism design packages.) Fig.3.7b shows a possible instantiation of a grouping class, in which group (type) names are non-unique attribute values. From the discussion of this example, the reasons for non-unique group names as attribute values can be concluded. Firstly, in terms of the semantic concept of the object convertibility described by Ter Bekke (1992a), values of all attributes of a class should be unique, instead of the value of a specific attribute. Secondly, in terms of the object identity
concept in object-oriented data models (Khoshafian & Copeland, 1986), instances of a class are determined by the system-generated object identifier, instead of any values of attributes which are meaningful to applications.

- A set-valued attribute could be thought of a grouping class from its underlying attribute class without consideration of any grouping expression.

3.3.4 Composite object
The nature of engineering objects in the machine design is of more complex structure than their quantity. Due to the limitation of the current state-of-art of the data modelling method a machine object is usually not represented as one instance in a specific class. An issue then arises:

\[
\text{how can several instances representing one physical object be manipulated as a whole?}
\]

This issue is called composite object representation; more recent studies are referred to (Dittrich et al., 1987; Kim, 1991; Liu, 1993; Ter Bekke, 1994; Chan et al., 1994; Zhang & Van der Werff, 1994c).

Fig. 3.8 shows two examples raised in the present study. In Fig. 3.8a, an engineering drawing, a kind of representation, of an assembly of mechanical objects is shown. At a more conceptual design level, 'component-1' is represented as two nodes in a global coordinate system (Fig.3.8b). If in a data representation schema, class 'NODE' is defined first, 'component-1' can then be viewed as a composite object which consists of two instance nodes from the node class. Clearly whenever the geometric information of 'component-1' is requested these two instance nodes must be accessed simultaneously. In Fig.3.8c bar-2 and gear-3 might become members of a composite object gear-3' when the length of bar-2 is less than the radius of gear-3. If in a data representation schema, two classes 'GEAR' and 'BAR' are defined first, gear-3' can then be viewed as a composite object which consists of two instances (a bar and a gear) from two classes, respectively. A CAD/CAM system is of general-purpose in the sense that more design cases or tasks can be supported. Therefore in the viewpoint of a CAD/CAM system more situations need to be considered. For example in an actual machine design there could be more nodes or even other types of geometric entities as elementary objects of 'component-1' (Fig.3.8b). For the case shown in Fig.3.8c there also exists the possibility that even when the length of bar-2 is less than the radius of gear-3 these two components may still be kept separate in the sense of the manufacturing and assembly design. (If so, gear-3' never exists.) The decision is completely in the mind of the designer - one of the database users.

From the above discussion, an important feature of the composite object in CAD/CAM of machines can be concluded, namely:
production of a composite object (including the information such as types and numbers of elementary objects) may be dependent on the database user, at the level of the instantiation of a database.

The data representation of composite objects must specify:

- which entity is considered as a composite object and which elementary objects it is composed of (elementary objects may themselves be composite objects in other contexts).
- elementary objects are part of a composite object so that they can be manipulated as a whole. This is usually called specification of a so-called part-of semantics (A is part-of B).

A comprehensive survey of various data representations for composite objects in the literature has been made. Four contexts to define a composite object are identified:

- object-oriented data models, e.g. (Kim, 1991),
For the case of Fig. 3.8b
(make-class frame
  :attribute(fixed-node-1: domain node
    composite true)
  :attribute(fixed-node-2: domain node
    composite true)
  ...........
)

For the case of Fig. 3.8c
(make-class gear-3'
  :attribute(gear: domain gear
    composite true)
  :attribute(bar: domain bar
    composite true)
  ...........
)

Diagram representation

Note:
The bold face string is keyword used in the syntax of building blocks

Figure 3.9. Two Representatives of Data Representation for Composite Objects

- semantic data models, e.g. (Dittrich et al., 1987; Liu, 1993).
- extended relational data models by Deppisch (1987) & Barsalou & Wiederhold (1990), and
- conventional relational data model, e.g. (Lorie & Plouffe, 1983).

A common feature to specify the part-of relationship is somewhat based on has-a connection. For the examples shown in Fig. 3.8, the definition for composite objects is shown in Fig. 3.9a, where the syntax by Kim (1991) is used. The method used by Dittrich et al. (1987) is slightly different from Kim's, as shown in Fig. 3.9b, in that his starting point seems to think that a composite object is somewhat to 'contain' several elementary objects as one can see from their 'envelope-like' representation diagram. The equivalence of the two methods is, however, not difficult to be found by considering 'STRUCTURE-IS' as an alternative means to express the part-of semantics based on has-a link. All these methods in the literature actually share the same instance pattern (Fig. 3.9c).

The current methods are suitable for the situation where the type and the number of elementary objects must be known to the database schema designer. The application in the
CAD/CAM of machine requires frequent changes of database schema. A new building block for the composite object representation is proposed, which can alleviate such changes.

The proposal takes the grouping data abstraction as a basis and the part-of relationship is built upon the member-of link. Both the representation diagram and textual syntax are shown in Fig.3.10a. (The textual syntax is adapted from Dittrich's STRUCTURE-IS constructor.) For the example of Fig.3.8b, the definition by this proposed building block is shown in Fig.3.11a. Since the instance of 'component-1' is a class of nodes which meet the grouping criterion, the schema definition is independent of the number of non-moving nodes with respect to the global coordinate system (see Fig.3.8b). With combination of is-a link the proposed building block may enable the schema definition to be independent of number of types of elementary objects composing a composite object. This can be shown with the example of Fig.3.8c. First, a generic type 'BODY' is defined (Fig.3.11b). Types like 'bar' and 'gear', etc. are defined as a subtype of 'BODY'. A composite object 'COMPOSITE-BODY' is then defined via the grouping shown in Fig.3.11c. The 'gear-bar' composite body is an instance of COMPOSITE-BODY' (Fig.3.11d). When a grouping expression is changed, various types of physical composite bodies can all be represented with this schema. To avoid the multi-grouping, a composite object can also be defined by following the pattern as shown in Fig.3.10b, where the grouping implicitly plays a role in the formulation of a composite object. The instance
pattern of a composite object by following the proposed notion is generally shown in Fig.3.10c.

![Diagram of object frame and composite body representation](image)

**Figure 3.11. Two Examples of Composite Object Representation Using Proposed Method**

### 3.3.5 Several guidelines for the conceptual data modelling

**Guideline-1 - meta data modelling.**

The idea in defining a composite object with combination of member-of and is-a links can be generalized as a means to realize a schema pattern shown in Fig.3.12a. In such a pattern, properties of an object in the real world are represented at the instance level. Usually the definition of properties of an object in the real world corresponds to the database schema (type) definition as the responsibility of the database designer, while instances of a class (of a certain type) are maintained by the database user. The scheme outlined in Fig.3.12a, though rather rough, exhibits an advantage that change of the property list of an object in the real world does not require the change of the schema definition. By intuition this scheme would be something as shown in Fig.3.12b. The scheme loses two kinds of information:

- the meaning of all attributes, and
- the meaning of all instances; both are of interest to a specific application.

Modelling of these two pieces of the lost information leads to some extensions (to that schema), as shown in Fig.3.12c. The schema shown in Fig.3.12c is basically similar to the meta model concept by Ter Bekke (1992a), but different in the starting points of view of the
Figure 3.12. A General Principle in Use of Data Modelling Building Blocks

problem. (The starting point of Ter Bekke (1992) is to reduce the number of tables for the same amount of application information.) Zhang & Van der Werff (1993a) pointed out that this schema generally requires more attention of the database user in performing operations on the database. They further advised to divide a set of properties of an object in the real world into the invariant and non-invariant ones with respect to an application-related process,
e.g. the machine design process in the context of the present study. Invariant properties are better defined at the type level while variant properties could considered to be defined at the instance level by following the schema shown in Fig.3.12c. In data modelling practice, it is possible to have some variations of the schema described in Fig.3.12c (a case to apply this guideline can be found in Chapter 5).

![Diagram](image)

*Figure 3.13. Grouping versus Generalization*

**Guideline-2 - grouping versus generalization.**

Grouping data abstraction is different from generalization data abstraction. Philosophically generalization is based on the analysis of a set of types of properties of objects in the real world. The set of common property types forms a new object type, i.e. generic object, see Fig.3.13a. Grouping is based on the analysis of instances of one or more classes and collects those instances, together into a class--grouping class, whose instance values meet a defined criterion at the schema definition level as well as at the instantiation level, see Fig.3.13b. In some semantic data models (Smith & Smith, 1977; Hammer & McLeod, 1981) at the super-type definition level a special attribute say 'type' is also defined to record, for each individual instance in the super-class, what a type of sub-class the instance belongs to. In such a case one might view the formulation of a specific sub-class as somehow via grouping, see Fig.3.13c. This is, however, not an appropriate interpretation because in a grouping class extra attributes (if any) are acting on a class of instances instead of on each instance as generalization does. Definition of a sub-class may also include predicates - a kind of set membership criteria (Hammer & McLeod, 1981), but the presence of these predicates is not necessarily meant for grouping. (In OO data models such a special attribute is absent.) In general, generalization is an abstraction of a set of objects based on their *property type*, while grouping is an abstraction of the underlying class based on particular *property values* of instances of that class.

**Guideline-3 - grouping versus aggregation.**
The grouping data abstraction is different from the aggregation data abstraction. Peckham & Maryanski (1988) made the following remarks:

Although grouping and aggregation define new object types from previously defined types, they represent fundamentally distinct abstractions. Aggregation provides a mean for specifying the attributes of a new object type, whereas grouping is the mechanism for defining a type whose value will be a set of objects of a particular type.

When extra attributes are defined with a grouping class and the grouping criterion is completely left for the database user's responsibility a member-of link might be thought of as a means to define a set-valued attribute of a new class.

![Diagram](image)

Figure 3.14. Object Relativity Principle versus Set-Valued Attribute

**Guideline-4 - the object relativity principle.**

In data modelling the building blocks to define data are class, attribute, relationship, etc. They carry the meaning of data. The object relativity principle in the data modelling is defined by Smith & Smith (1978):

An object can have various interpretations for different abstractions. Relationship, class or entity, attribute, category, attribute value and instance are just different interpretations of the same abstract object.

In other words, any schema constructor with guidelines meeting the object relativity principle must allow applications to view the same data as class, attribute, relationship, etc.

In the guideline by Smith & Smith (1978) for using the object relativity principle in the conceptual data modelling, a necessary condition is imposed on whether or not an object can be viewed as an attribute of another object:

If an object A is viewed as an attribute of an object B, the functional correspondence (from B to A) must be n:1. —a necessary condition

The condition does not fully agree with the object relativity principle as it may limit some
meaningful views of object A as an attribute of object B. For example it does not allow, for the application semantics as shown in Fig.3.14a, parts to be viewed as an attribute of a product (Fig.3.14b) because the above condition is invalid. (Only a product can be viewed as an attribute of a part, see Fig.3.14c). The main reason for the introduction of the above condition is lack of the notion of the set-valued attribute for the conceptual data modelling or in other words the above condition is only true in the relational environment. With introducing the set-valued attribute notion 'part' can be viewed an attribute of 'product' (Fig.3.14b).

![Diagram](image)

**Figure 3.15. Limitation in Data Representation due to Lack of Object Relativity Principle**

Kaper (1987) established a relational model of a catalogue of mechanisms for some queries, e.g. given a mechanism id to find the structure information of this mechanism. He needed to describe both individual mechanism structures and a catalogue structure. Obviously these two structures are quite different. Because the version of a relational DBMS (ORACLE) he used does not allow an attribute to be viewed as a class in another view-context, he could not define a schema roughly shown in Fig.3.15a, where the mechanism is a type and the mechanism catalogue is another type. His idea was to define several tables using 'mechanism-id' and mechanism-body-id' as a key attribute (Fig.3.15b). All mechanisms with their components are uniformly stored in these tables. (Fig.3.15b only shows one of many such tables where 'mechanism-id' and 'body-id' are taken as key attributes.) To get the structure
information of a given mechanism searching and joining of these tables in terms of the given 'mechanism id' is necessary and this causes a lot of computing time. Consequently an undesirable performance with very low data access speed was gained. The underlying reason of this problem now becomes obvious, i.e. 'mechanism' cannot be viewed as a class in one context while in another context as an attribute.

Guideline-5 - A better application database model.
Using semantic data models or object-oriented data models does not guarantee that a better application database model will always be established (Zhang & Van der Werff, 1993a). General guidelines towards a better application database model will be helpful. To be a better model, the following points are of importance.

- The logical data dependency among attributes of a class should be completely removed. For example, in a 'student' class, defining both 'department' and 'department-address' as attributes of 'student' class should be prohibited.
- The data redundancy should be controlled at minimal. The controlled data redundancy usually means that the data dependency can be specified in mathematic formulae.
- The data integrity should be at best maintained with the internal constraint of data structures that a particular data model supports. Any method for the external constraint for the data integrity requires some extra effort from the DBMS side.

3.4 The Database Approach to CIMOME Development

Designing of data structures usually takes place at the coding level. This requires much expertise of programmers on both the understanding of application semantics and building blocks of an individual programming language. Data modelling (in particular hereafter meant for the conceptual data modelling), as one of the disciplines in computer science, is aimed at helping system developers to put their attentions on the modelling of application semantics. (The capability of data modelling was demonstrated in the previous sections of this chapter.) This is a general reason for the idea: replace the data structure design with data modelling. More precisely, data modelling at least provides a means to design data structures at the conceptual level where principles of data structures are developed (with analogue to the conceptual design for machines). Halbert & O'Brien (1987) implied this point with the following comment (when elaborating on the relationship between data modelling and object-oriented programming language):

Basically, the addition of database abstractions to object-oriented programming provides a conceptual separation between modelling source code and implementing source code, but identifying which is which examining code is still difficult for the reader, because they are inseparably intertwined within the same class structure.
In the context of integrating several separate programs (in the present study, tools for the designing and manufacturing of mechanisms), data modelling plays a particular role. Modelling of the same application semantics using a computing language (e.g. FORTRAN) might lead to quite different syntaxes dependent on different programmers. This is particularly true when some lower-level computing languages are used, as illustrated in Fig. 3.16a. (More recently such an observation has also been made by Zimmermann et al. (1992) and has led to a research issue called object-oriented finite element programming.) Furthermore, such diversity will be increased in an integrated environment program system, because different tools (programs) essentially handle different application semantics. The diversity in syntax obviously introduces an extra difficulty for integration of these tools programs. Data modelling thus provides a relatively unified syntax with respect to these tools programs (Fig. 3.16b); it is a kind of protocol among tools programs.

![Diversity in Syntax in Data Representation versus Computer Languages](image-url)

**Figure 3.16. Diversity in Syntax in Data Representation versus Computer Languages**

Using data modelling to the system development two problems are produced (hereafter these two problems are referred as to *database problem*):

- the incompatibility between a database language (data definition and data manipulation) and a computing programming language. An extra complexity of the interface between these two cultures arises, which itself is devoted to more research (Sim, 1992).
- low data access speed with respect to the need of engineering applications. (Usually DBMS handles a large amount of persistent data in the secondary memory.)

These two problems must be considered at the implementation level where a specific DBMS and a host computing language are considered.

The current implementation of CIMOME considers the integration of tools for the mechanism conceptual design. (These tools have been introduced in Chapter 2.) All these tools were written in FORTRAN and they are of computational nature with non-well developed user-interface (see also the discussion of Section 2.4.1). It was not yet justified that commercial
DBMSs (including object-oriented ones) are suitable to the requirement of the CIMOME implementation (also see Chapter 2, Section 2.5). Furthermore FORTRAN provides nothing but ARRAY constructor for data structures. Syntactic diversity may lead to unwieldy source codes. The solution taken is thus to develop a main memory database management system called ABC system (Langbroek, 1989; Zhang, 1993). (In Chapter 2, Section 2.5 there are some reasons for developing the ABC system.) Considering the above mentioned database problems, ABC is written in FORTRAN so that the first database problem disappeared. The feature of main memory database management systems is that algorithms for data searching and sorting are developed in the context of the internal memory. The data persistence is simply done via the unloading of memory databases (Fig.3.17). While ABC is used for the application program development, the developer’s attention is put on the representation of application semantics, instead of whether a specific portion of data should be persistent or temporary. Usually, ABC is responsible for the description of the structural part of an application, while a host program takes care of the behavioural part.

3.5 Features of ABC system

Virtual space
ABC is based on a virtual space in the main memory, where all instances of a current database are stored. At the implementation in FORTRAN the virtual space is defined as a two dimensional array. The size of the virtual space can be easily changed dependent on the size of the internal memory reserved. Although this may limit the number of available objects, Van der Weerd (1992) argued that transparent hardware solutions, where the distinction between objects in memory and objects on disk disappears, have already been realized. This implies that in future distinction between the memory and disk may not be necessary.

Object identification
ABC system allows for either identification or non-identification of objects. The latter purely corresponds to the record type of data structures in PASCAL and it is actually the matter that
the database user is responsible for object identification. Two methods for the object identification are provided in ABC. The first is that the system provides an identifier for each instance such that the identifier does not encode any meaning of the object. The second is that ABC system allows for the specification of certain attributes as unique such that a set of values of these attributes is served as an identifier of instances. The system object id is composed of two aspects: class id and instance id, i.e. system-generated object id = \(<\text{class-id, instance-id}>\).

**Figure 3.18. Data Storage Organization of ABC**

**Data storage organization**
The most significant design decision made for the ABC system is to concentrate on meeting the needs of interactive engineering applications, e.g. CIMOME. For this reason, ABC has a relatively sophisticated main memory implementation and a simple secondary memory implementation. In the typical ABC application, a number of objects are read into the main memory at the start of an engineering session, edited for an extended period of time, and written back out to the secondary memory at the end of the session. Fig.3.18 illustrates the storage organization of ABC. Secondary memory organization is an aggregation of the main memory organization in the sense that in the secondary memory, the objects can be managed by a single file, and in the main memory, the objects are separated in individual tables. Tables are automatically indexed by ABC, when in the main memory, for fast access.

**Marking raw data**
In each cell of attribute values the ABC system provides a means to mark whether the data in the cell is defined or not with a global flag variable. This provides a facility for supporting

---

3 More details are referred to Zhang (1993g) about how ABC system is used as a means for FORTRAN to gain support of well-known data structure types of record and pointer.
raw data handling (see Chapter 5).

**Database schema evolution**

This feature indicates that at run-time database schema definition can be modified. When ABC is used only as a tool for FORTRAN to gain support for the data types of record in PASCAL, this feature implies that the record type definition can be modified at run-time, which is, however, not supported in most of the modern computing languages.

3.6 Discussion and Conclusions

The above discussion has shown the so called *database approach* to the CIMOME realization. The research to the CIMOME development exposes some week points in the current conceptual database modelling discipline with respect to the CAD/CAM of machines. Grouping data abstraction has been thoroughly discussed here and has shown its promising in a proposed data representation of composite objects. In the database design *guideline-1* an idea is formulated, that is, how to define a database schema which is flexible to frequent changes of objects in the real world in their quality (i.e. object type). The meta model proposed by Ter Bekke (1992a) can be derived from this idea and has been made more explicit for use. The idea actually founds the nature to cope with the relationship between "static" and "dynamic" modelling world. (A detailed elaboration is beyond the scope of the present study.) Whereas the database research was directed to look for a new DBMS which can facilitate the type evolution (Zdonik, 1990) the idea presented here shows an alternative way to cope with the requirement of frequent changes of object type. In *guideline-4* a necessary condition previously set by Smith & Smith (1978) has been removed to promote more object relativity—an application-oriented attitude to data modelling. In subsequent chapters the application of data (database) modelling introduced in this chapter will be demonstrated.
The Mechanism Development Model

4.1 Introduction

Products are materialized, artificially generated objects or groups of objects which form a functional unit (Krause, et al., 1993). In terms of this definition, a mechanism is considered as a kind of product. Therefore the term mechanism is interchangeably used with the term product hereafter. This will be helpful to utilize the achievements of the research on the general product modelling to the mechanism development.

In Chapter 2 the CIMOME architecture has been established and the functions of major system components are described. The current chapter initiates the realization of these system components and makes some contexts for the discussions of the subsequent chapters concerning the realization of CIMOME. Since any computer system is essentially to simulate partial human (also including machines, e.g. NC machine) activities on the realization of a physical object, system components are models of these activities. Establishment of these models is the key to the realization of these system components. So this chapter will concentrate on the discussion of these models. In particular, first the basic concepts of model, product model, process model and product development model are clarified (Section 4.2). A general framework of product model is introduced in Section 4.3, where the development strategy of a product model for mechanisms is also illustrated. In Section 4.4 a requirement model for mechanisms (a partial product model) which is considered for the current implementation of CIMOME is introduced. This can be viewed as an extended discussion of Section 2.2.3 (Chapter 2), where several mechanism design problems are outlined. The information of this model is stored in the data repository and managed by the data manager described in the CIMOME architecture (Chapter 2). In Section 4.5 a design process model (a partial process model) for mechanisms which is considered for the current implementation
of CIMOME is introduced. The process model is used by the *process manager* described in
the CIMOME architecture (Chapter 2). Section 4.6 concludes this chapter.

4.2 Model Concept

A *Model* is a representation of physical objects (e.g. organizations, natural phenomena, etc.). Generally speaking a model makes the observation of its counterpart easier. (A counterpart may not exist at all, e.g. product to be designed.) In terms of various types of representation means and types of objects, models are classified. *Physical models* represent some natural phenomena so that some famous physics laws were discovered. Observations of behaviours of products usually need a *mathematic model* which represents the relationship between *cause* and *effect* in terms of internal constraints of products. The perception of cause and effect relies on the interaction of products with their environment. A well known example in the machine product development is the *input-output* motion relationship which can usually be represented by a set of mathematic equations. The *Engineering drawing* with its attached comments made by designers is a kind of model of physical counterpart structures with designers' intents. When any model is digitized in the computer the model is called an *application data model* or *computer model*. The model below usually refers to this kind.

A *product model* is a set of models which represent a product from various aspects: its **structures**, **requirements**, **behaviours** and **functions**. For example, *product structures* are viewed in the context that a product is a geometric entity, while *product requirements* are viewed in the context that a product has to fulfil certain needs to the human being. A *process model* represents a *process chain* which is usually performed by human product developers together with some computer programs (e.g. the so-called *service tool* defined earlier in Section 2.4.1 of Chapter 2). A *process chain* contains a set of tasks or actions which make decisions on the product structures as well as on the ways a product is manufactured and assembled. Such decisions need knowledge (e.g. technological data, etc.) and previous decision courses which are also represented in the computer as a kind of model. Fig.4.1 exhibits their relationship. A *product development model* embodies all these models.

In the sense of the computer-aided product development, there are two kinds of users: the computer *system developer* and the *product developer*. The former develops the model schema while the latter develops the model data or instances in conformity with a defined model schema. In certain occasions the product developer may also create the model schema. When this happens there must be a *meta model schema* available (the *meta model concept* is discussed earlier in Section 3.3.5 of Chapter 3). *Product modelling* is a process to create a product model. Two kinds of users play different roles in the product modelling, respectively. The system developer is responsible for defining all types of the model schema such as the product model schema, the process model schema and the design knowledge model schema,
etc. The system developer is also responsible for creating instances of the process model and the design knowledge model, while the product developer is responsible for creating product model data which correspond to an individual product. The research goal of the system developer in the product modelling should be to define a general model schema for various types of products and/or guidelines for developing the model schema for specialized products.

![Diagram of product development model concept]

*Figure 4.1. Product Development Model Concept*

### 4.3 A General Framework of the Product Model

A product model contains many sub-models as mentioned in the above discussion, e.g. the models for structures, requirements, etc. Further, with respect to the various product development stages (see Section 2.2 of Chapter 2) the information is decomposed into layers. Each layer corresponds to one model (sub-model). Since sub-models are related, any change on one sub-model may influence others. It is important to describe those relationships among models. More generally the term *link* is used for the *relationship*. Such a link is more than a relationship in the sense that it also includes the *context* information of a view through the relationship as well as the *method* to maintain a relationship.

Types of Links between sub-models depend on types of products considered. For example, to the mechanism product the model of *marketing* may not make much sense; no any link is
needed to relate to the marketing. There are, however, some generic links which are worth to be investigated. The first generic link is concerned with object and object-view. This link says that in a model space some information blocks may be views (termed object-view) of another information block (termed object) in a ceratin context. When an information block A is view of B, the information block A can usually be derived when the A view of B link is defined. When any change takes place in information block B, influences on information block A can then be automatically maintained. Fig.4.2 shows that a kinematic mechanism model can be viewed as a graph when only its topology is concerned. The other generic link is that information block A is to add some more information on information block B. For instance the information block of the mechanism as shown in Fig.4.3b is to add volumetric information on the information block as shown in Fig.4.3a. (Definition on these two information blocks can be found in Chapter 5.)

From the above discussion a general framework of the product model, which consists of two types of links, can be envisaged as shown in Fig.4.4. Several remarks on this figure are made below.

- In terms of the object-orientation, the information about product structures is at the bottom layer of this framework. This implies that the information about product structures is considered as primitive while other information is viewed of that.
- Product structures have different layers corresponding to design stages. The information block from later design stages is considered specialized of the information block at earlier design stage.
- Product structures can be viewed in different contexts. For the example of a lamp product

66
Section 4.3  A General Framework of Product Model

Figure 4.3. A Case of Specialized-of Link

Figure 4.4. A General Framework of Product Model

Figure 4.5. Different Views of Product Structures

Fig.4.5 shows two views of the lamp: one is in the context of the so-called part-of (i.e. A is part of B) (Fig.4.5a) and the other is in the context of the functionality of the lamp (or working principle of the lamp) (Fig.4.5b). Neither of these two views is the product structure itself. The product structure itself is something that is represented in the
engineering drawing. (Notice that the information represented by the *engineering drawing* is not only geometric but also non-geometric.)

- In Fig.4.4, two information blocks 4 and 5 are *specialized* of the same information block 3. This implies that a well known design practice, i.e. different versions of a design, is included in this framework.

- Two information blocks 2 and 9 are viewed of the same information block 3. This implies that a well known design philosophy, i.e. different views of the same object is included in this framework.

- An information block 2 which is a *view of* information block 3 can itself be a source of other views (e.g. information block 1). This follows from the *information relativity principle* (see Appendix-A).

- There are two kinds of object views in terms of their purposes. The first kind of object view is for generating product behaviours, e.g. a structure is viewed as a set of beam elements for generating its strength and stiffness behaviours. The second kind of object view is for facilitating the design, e.g. the functionality view of a lamp (Fig.4.5b).

- The label on each link is an identifier for the reference to the information, e.g. contexts of a link, purposes of a view, methods to maintain this link, etc.

![Figure 4.6. The Relationship between Tools and Product Model](image)

The discussion of the framework above implies that for the CIMOME realization, the model for product structures should be developed first (see the next chapter). Each design tool introduced in Chapter 2 is considered working on a certain sub-model which is then brought into the product model under the above described framework with a corresponding tool being brought into CIMOME. A product model is in incremental growth with the CIMOME growth. This is the basic strategy of the CIMOME realization. The relationship between tools and a product model can be naturally derived, as shown in Fig.4.6.
4.4 A Partial Requirement Model for the Mechanism Design

The requirement for the mechanism design consists of required behaviours and required constraints as defined earlier in Section 2.2 of Chapter 2. In the following a partial requirement model is outlined which is currently being used in the implementation of CIMOME. In particular the discussion is focused on the data representation of required behaviour functions. First, semantics of required behaviour functions are described and a model (more precisely, a conceptual database model schema) is then defined.

![Diagram of mechanism required behaviour function, path generator, and motion generator](image)

Note: $\alpha = \text{input motion variable}; (x_i,y_i)$ is coordinate of a point on a path

*Figure 4.7. Required Behaviour Function Types in the Mechanism Design*

4.4.1 Semantics of required behaviour functions

Required behaviour functions in the mechanism design can usually be expressed in an *input-output behaviour function* which can generally be expressed as: $y = f(x)$. Considering the mechanism with one Degree Of Freedom (DOF) required behaviours are usually classified into three well-known types as shown in Fig.4.7a. From each type, more types could be produced. For example, a path generator design problem can further be classified into two problems in terms of whether required points on a path are also requested to correspond to specific values of an input motion variable, see Fig.4.7b. Required behaviour functions also include the input and output motion types, e.g. full rotation, reciprocating motion, etc., as mentioned earlier in Section 2.2.3 of Chapter 2.

A required behaviour function itself, i.e. $f()$ can generally take two expression forms: *discrete* (i.e. $<x_i,y_i>$, $i=1,2,...,n$) and *continuous* (e.g. $y=a()+b$, where $a$ and $b$ are parameters). The characteristic of a required behaviour function is reflected in $f()$ itself and is independent of the expression forms when errors resulting from numerical calculations are neglected. The combination of different $f()$ with perhaps different expression forms as one overall required behaviour function is also possible, see Fig.4.7c,d, where the expression form of $f_i()$ ($i=1,2,...,n$) can be either discrete or continuous.
Required behaviour functions may also relate to the first (or higher) order derivative of the function, for which the motion quantity (MQ for short) will be called. For example, \( y' \bigg|_{x=x_1} = c_1 \) (here \( y' \) is a kind of MQ; \( c_1 \) is a constant). A formal statement of a design problem related to motion quantity would be: to find a mechanism which is satisfied with the following condition (motion condition for short):

**Expression-1:** \( BF_{\text{actual}}(MQ_1, MQ_2, MQ_3, \ldots, MQ_n) = V_1 \)

**Expression-2:** \( BF_{\text{actual}}(MQ_1, MQ_2, MQ_3, \ldots, MQ_n) = V_2 \)

**Expression-m:** \( BF_{\text{actual}}(MQ_1, MQ_2, MQ_3, \ldots, MQ_n) = V_m \)

where, \( m=\) number of motion conditions; \( n=\) number of motion quantities; \( BF_{\text{actual}}() \) is an actual behaviour function; \( V_i=\) required values. For instance, an expression would be: \( y \bigg|_{x=x_0} = 3.5 \). The above general discussion implies that an expression for a motion condition could also be composed of more than one motion quantity, e.g. the expression: \( y_{\text{max}} - y_{\text{min}} = 30 \), where both \( y_{\text{max}} \) and \( y_{\text{min}} \) are motion quantities. (This is a well known problem called stroke generator.)

**Function features**

<table>
<thead>
<tr>
<th>Feature graphical representation</th>
<th>Mathematical expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dwell motion</td>
<td>( y = y_1 = y_2 = \text{constant} )</td>
</tr>
<tr>
<td>Constant velocity</td>
<td>( y = y_1 - y' (x-x_1) )</td>
</tr>
<tr>
<td>Stroke</td>
<td>( y'(x=x_1) = 0, \ y(x-x_1) = y_1 ), ( y'(x-x_2) = 0, \ y(x-x_2) = y_2 ), (</td>
</tr>
</tbody>
</table>

*Figure 4.8. Behaviours Function Feature Classification*

Although required behaviour functions can be expressed in a formal way as described above a CAD/CAM system would be much more intelligent if required behaviour functions can be expressed by the end-user (the product developer) with their domain specific terminology. The underlying philosophy of this idea is completely similar to that behind feature-based solid
modelling (Bronsvoort and Jansen, 1993). An analogy exists as shown below:

<table>
<thead>
<tr>
<th>behaviour function modelling</th>
<th>solid modelling</th>
</tr>
</thead>
<tbody>
<tr>
<td>level-1 function features</td>
<td>&lt;&gt; form features</td>
</tr>
<tr>
<td>level-2 motion quantity models</td>
<td>&lt;&gt; solid models</td>
</tr>
</tbody>
</table>

Fig. 4.8 lists some features of behaviour functions well known in the mechanism design. For the design problem *stroke*, on level-1 the problem is simply stated as finding a mechanism with output stroke = 30 (mm); on level-2 the same problem is expressed as the following motion quantity model: \( y' \big|_{x_1} = 0, \ y' \big|_{x_2} = 0, \ y_1 = y \big|_{x_1}, \ y_2 = y \big|_{x_2} \) and \(|y_2 - y_1| = 30.\)

**Figure 4.9. Definition of Class 'VARIABLE'**

### 4.4.2 A conceptual model schema of required behaviour functions

In the following the data representation of the behaviour function for the function generator problem (see Fig. 4.7a) is focused. The key points to develop a conceptual model schema of behaviour functions are first summarized below:

- The model is a collection of types of classes, and a particular design problem corresponds to the instantiated model. In this sense the model is generic and called Generic Function Model (GFM for short).
- The description of a function (in mathematic sense) consists of two parts: the description of function variables (e.g. 'x', 'y' in the function \( y = f(x) \)) and the description of the relationship between variables (i.e. \( f(x) \)).
- The concept called *atomic function* and *molecular function* is introduced. When in a defined region of a domain of active variables there is more than one type of function
defined, each sub-function is atomic and combination of them is therefore molecular. For example, in Fig.4.7c,d, \( f_i(x) \) \( (i=1,2,\ldots,n) \) is atomic. The underlying philosophy is similar with the idea in describing product: a product is an assembly and an assembly contains parts. An analogy exists as shown below:

<table>
<thead>
<tr>
<th>function</th>
<th>product</th>
</tr>
</thead>
<tbody>
<tr>
<td>atomic function</td>
<td>part</td>
</tr>
<tr>
<td>molecular function</td>
<td>assembly</td>
</tr>
<tr>
<td>overall behaviour function</td>
<td>product</td>
</tr>
</tbody>
</table>

Fig.4.9 shows a definition for a class 'VARIABLE'. Several remarks are made below.

- When the value of attribute 'motion-order' is '1' it means that only the first order motion quantity is recorded. This is intended for modelling e.g. the problem constant velocity (Fig.4.8) where possibly the zero order motion quantity \( y_1 \) and \( y_2 \) are not of interest to the design problem and thus not predefined. In general it is intended for the representation of the problem given directly by the first (or higher) order motion quantity.

- When the value of attribute 'time-concerned' is 'yes', the first order (or more) motion quantity is derived with respect to the time, e.g. the first order motion quantity is velocity.

Fig.4.10 shows a definition for a class 'ATOMIC-FUNCTION'. Several remarks are made below.

- The value of attribute 'motion-order' should be consistent with that in the class 'dependent-variable' because dependent variable is determined by the function. An external constraint rule shall be established as shown below (notice that attribute 'dependent-variable' is itself a class of type 'VARIABLE'):

  \[ GFM \text{ External Constraint rule } <1> : \]

  value of 'ATOMIC-FUNCTION.motion-order' =
  value of 'dependent-variable.motion-order'.

- The schema implies the view that the discrete expression form is common to all continuous expression forms, i.e. various continuous representation forms are considered as specialization of the discrete form. In fact, a required behaviour function is in most cases first given in a discrete form \( <x_i,y_i> \) and then may be converted into the representation of some continuous forms, e.g. Fourier Series in the system called TADSOL (Rankers et al., 1985). The discrete representation serves control points for those continuous representations (e.g. Fourier Series). In case an object-oriented programming system is considered for the implementation, the method to convert from a discrete form to Fourier
Series form should be defined at the sub-class level 'Fourier-Series' (Fig.4.10).

- The schema can be used for both the required behaviour functions and the actual behaviour functions.

```
[Diagram: VARIABLE \rightarrow ATOMIC-FUNCTION
  \rightarrow independent-variable \rightarrow dependent-variable
  \rightarrow function-name \rightarrow required-or-actual \rightarrow function-type
  \rightarrow Fourier-Series \rightarrow 1-order-polynomial

  number-of-harmonic-order \rightarrow [A B C \Phi \ P1 \ P0]

  motion-order \rightarrow [task-id]

  Domain('function-type')=('Fourier-Series', '1-order-polynomial', ...)
```

*Note: The meaning of attributes 'required-or-actual' and 'motion-order' are same as those in definition of class 'VARIABLE'.*

**Figure 4.10: Definition of Class 'ATOMIC-FUNCTION'**

Fig.4.11 shows the definition for a class 'BEHAVIOUR-FUNCTION' for the function generator. Several remarks are made below.

- An overall behaviour function is composite in that it may consist of more than one sub-functions as mentioned earlier in Section 4.4.1. Thus it is defined as a composite object from class 'ATOMIC-FUNCTION' and has its own attributes.

- The information represented in class 'ATOMIC-FUNCTION' is actually viewed as common to sub-classes 'DWELL', 'CONSTANT VELOCITY' and 'STROKE', etc. This is for two reasons. *Firstly*, these problems, e.g. 'dwell', 'constant-velocity', etc. are essentially a function generator. They can inherit information from 'ATOMIC-FUNCTION', such as the information about corresponding variables. *Secondly*, the information about themselves may need classes 'ATOMIC-FUNCTION'. For example, for the problem of the dwell motion function generator, the behaviour functions beyond the dwell region need to be described and the dwell motion itself can be represented as a horizontal line such as y=constant. This also implies that the consistence between class 'ATOMIC-FUNCTION' and these sub-classes such as 'dwell' should be maintained.

**4.4.3 Introduction to an implementation: EDFUNC**

The Generic Function Model (GFM) is adapted to an implementation of a system called EDFUNC (EDit FUNCTION). The system is a user-interface based upon GFM. The system is implemented using the ABC system (Chapter 3) and a User-Interface Management System
(UIMS) (Chapter 7), both of which are developed in parallel with the present study at our laboratory. To give an impression, Fig.4.12 shows two windows of EDFUNC. The system allows the designer to specify a required behaviour function, such as a path, an input/output function, etc. (see Window-1). As regard to the function, representation forms can be discrete or continuous, periodic or non-periodic, etc. (see Window-1). Window-2 shows that function segments are specified. One is the standard function termed 'cycloidal motion' (the input motion variable: 0-30). The other is 'ith' degree polynomial function' which is now used to specify a dwell motion (the input motion variable: 30-60).

4.5 A Partial Process Model for the Mechanism Design

In Chapter 2 it is highly recommended that CIMOME takes a more advanced approach to the mechanism product development, that is, explicitly to represent development activities so that a development process is co-driven by the system and the end-user (the product developer). To achieve this goal the key mechanisms as implied in the discussion of Section 2.3.3 of Chapter 2 are a product process model (PPM for short), a process manager and an interface for the product developer. In the viewpoint of the product development process management, the CIMOME architecture can be displayed as shown in Fig.4.13. (With the similar idea
Figure 4.12. A User-interface Based Upon GFM
introduced earlier in Section 2.4.2 of Chapter 2 (Fig. 2.10b) this view of the architecture is also applied to each design tool level.)

![Diagram showing the CIMOME Architecture in Viewpoint of Process Management](image)

*Figure 4.13. CIMOME Architecture in Viewpoint of Process Management*

PPM should represent relationships among tools, a product model and development activities. Development activities are in general composed of a set of tasks which are further composed of a set of actions. Division of task and action follows the *information relativity principle* (Appendix-A) in that an action can be viewed as a task when an action is composed of a set of actions too. An actual development process is simply a traversal of a portion of tasks and actions. (This is geared to the discussion about the design process in Section 2.2 of Chapter 2.) So a data representation of development activities should express that

- a task contains a set of actions \( (a_1, a_2, \ldots, a_N) \) (Fig. 4.14a),
- tasks are executed in a predefined chain (Fig. 4.14b).

Fig. 4.14b also shows that a task can be fulfilled by more than one process chain.

Each action does something on a product model. An action can be activated only if

1. its predecessor action has been done, and
2. some resources, which are referenced to the product model data, are available (so-called *required resources*).

Each action also results in some resources which are referenced to the product model data (so-called *provided resources*). In general, the status of an action or task can have four states:

- NOT-READY (required resources are not prepared),
Section 4.5  A Partial Process Model for the Mechanism Design

- READY (required resources are prepared),
- ACTIVE (an action is doing), or
- DONE (an action is finished).

![Diagram](image)

*Figure 4.14. Task/Action and Task Execution*

To record the status of each action a *status variable* should be attached to each action or task. The status of a task is determined by the status of its actions. The state of a task or an action is therefore encoded in its status variable. A history of the task execution can be recorded by the status variables of all actions or tasks whenever any action is done.

A process manager enacts the process model, as mentioned earlier in Chapter 2 (*e.g.* Fig.2.10). Its tasks can be summarized with regard to the information that a process model describes above. The *first* task is to check whether required resources are ready for a task or action. The *second* task is to provide guidelines for the end-user about the current development state and the next step. The *third* task is to allocate required resources for a specific task or action, either from the end-user through the user-interface manager or from the current state of product data model. The *fourth* task is to update the status variable after each task or action done.

A conceptual data representation of PPM is developed and shown in Fig.4.15. The model is defined as a group of classes so that an individual design process with respect to a type of product is an instance of these classes. By doing so the model gains generality. Some remarks regarding this model are made below.

- Each action may contain more than one tool.
- *Required resources* and *provided resources* are dependent on a specific type of product developed and related to *states* of product model data. A class type 'PRODUCT-STATE' is defined in which relevant methods to inquire whether a specific state is reached is also specified. Attributes 'required-resource-state' and 'provided-resource-state' are classes of
type 'PRODUCT-STATE'. So the information about the availability of required resources can be inquired from the class object of 'PRODUCT-STATE'.

- Task execution chain is a directed graph expressed by class 'TASK-EXECUTION'. The first attribute holds a source-node in a directed graph and the second attribute encodes an outgrowth node.

![Diagram of TASK/ACTION and PRODUCT-STATE relationships]

*Figure 4.15. Data Representation of Process Model*

### 4.6 Discussion and Conclusions

Nowadays the term *model* is a buzzword. Because of the diversity of models used in various scientific and engineering domains it is necessary to clarify their meaning in order that the concept of product model, process model and product development model can be clearly defined (see Section 4.2). *Product modelling* is another buzzword. A recent survey of product modelling research by Krause, *et al.* (1993) does not distinguish different roles between a CAD/CAM system developer and a product developer who may use a CAD/CAM system. It is made clear in Section 4.2 that the role of a CAD/CAM system developer in the product modelling is to develop a general model schema which is not an actual product model of a specific product. The framework of the product model described in Section 4.3 has established a context of the discussion of the next chapter (Chapter 5) where a model of conceptual mechanism structures is developed.

The requirement model, in particular the required behaviour function model for the mechanism design, has been studied here. In Chapter 5 it will be shown how the required
Section 4.6

Discussion and Conclusions

behaviour function model is related to the model for conceptual mechanism structures. Use of the EDFUNC system in CIMOME will be described in Chapter 9.

As mentioned earlier in Chapter 2 the product development process management is relatively less discussed in the current CAD/CAM research domain. In developing CIMOME this has been observed and studied. The process model developed in this chapter has founded the process model in an implemented partial CIMOME system called CAMDES which will be discussed in Chapter 9.
A Database Model for the Conceptual Structure of Mechanisms

5.1 Introduction

In Chapter 4 a framework of the mechanism information model has been described, where the importance of the data representation of mechanism structures is clarified. Nowadays machines are being more complex and the requirement on development with the short lead time and low cost is increased. To cope with such a challenge a total development process must be decomposed into more stages, e.g. well-known stages: conceptual, embodiment and detail (Fig.1.2). This also enables the concurrent engineering\(^1\) to the product development. Another means to cope with this challenge is to use more computer-supports.

To have sophisticated computer support, the mechanism structure information must be represented in the computer. However, many studies have been devoted to the representation of the mechanism embodiment structure information which concerns quite a different design stage, and deals with details which are not required for the conceptual design. In other words the data representation of the conceptual structure of mechanisms is relatively missed. In Section 5.2 this standpoint will be elaborated in more detail. This founds the necessity of the study to be described in this chapter. In Section 5.3 requirements are set up on developing the data representation of the conceptual structure of mechanism with consideration of the intelligent support for the conceptual design of machines. Main ideas behind developing GMM are also illustrated. The output of Section 5.2 and Section 5.3 calls for a new view of

\(^1\) One of the natures in implementing the concurrent design is regarded to reorganizing a design process into smaller steps which might then be proceeded simultaneously. This is practised in developing the CIMOME architecture (see Chapter 2).
the conceptual mechanism which is called object-based view (section 5.4). The object-based view lays a semantic foundation for developing the application data model of the conceptual structure of mechanisms (in Section 5.5) using the knowledge of data modelling introduced in Chapter 3 (hereafter this model is called GMM for short). In Section 5.6 the link between GMM and assembly models which capture the mechanism information at the embodiment design stage is discussed. A preliminary evaluation of GMM against its use in CIMOME is briefly provided in Section 5.7 while details are left to subsequent individual chapters. A summary with discussion of related work is presented in Section 5.8 to highlight the salient points of the study presented in this chapter.

5.2 The Need of GMM for Intelligent Support to Conceptual Machine Design

5.2.1 Definitions
Definition 5.1. Features. Features are generally information sets that refer to aspects of form or other attributes of a part, such that these sets can be used in reasoning about the design, performance or manufacture of the part or assemblies they constitute (Shah, 1991). Shah & Rogers (1994) further extends the feature concept to assemblies, i.e. relationships between parts.

Definition 5.2. Structure. Structure is a representation of parts of a whole and their relationship to each other (NN, 1987). Since parts and their relationship can be viewed differently in terms of abstractions needed for a specific step in a total development process, there are conceptual structure, embodiment structure, etc. in the machine development domain. In terms of definition 5.1 the structure is therefore the representation of features.

Definition 5.3. Conceptual structure of mechanisms. Features of the conceptual structure of mechanisms are either of non-volumetric relevance such as the colour of a part, or indirect volumetric relevance such as the module of a gear. The information about a mechanism at this level is more concerned with the functionality of the mechanism, such as the kinematic motion. In the machine design some information at this level can be represented diagrammatically by the so-called the kinematic mechanism diagram (Fig.5.1b) and the geometry to describe this kinematic mechanism diagram is called kinematic parameter (Fig.5.1c). Fig.5.1a shows the embodiment mechanism structure where the feature information is about both volumetric parts and volumetric assemblies, i.e. the layout of volumetric parts in a bounded space.

Definition 5.4. Top-down and bottom-up design processes in the mechanism design domain. The top-down design process starts with the requirement specification, followed by the conceptual design and then the embodiment design. The bottom-up design process starts with the requirement specification, followed by the embodiment design. In the top-down
process the requirement specification will be a source to derive conceptual physical components as well as conceptual physical assemblies, while in the bottom-up design process the requirement is only used in the evaluation of the performance of selected volumetric physical components as well as volumetric physical assemblies. The bottom-up design process is referred to the situation in a real design practice that the designer selects solutions from his case library accumulated with his working experiences in design.

![Diagram](image)

**Figure 5.1. Embodiment and Conceptual Mechanism Structures**

5.2.2 The Role of GMM

The discussion about definition 5.3 implies that the conceptual structure of mechanisms is identified in the designer's world and belongs to one meaningful design stage. In the viewpoint that a perfect communication largely depends on whether two involved parties have the common knowledge on the information to be transferred, in the computer GMM is therefore needed. GMM is in a sense the knowledge representation about the conceptual structure of mechanisms in the computer's world. As a result, an one-to-one correspondence between the end-user and the computer over the machine design process is established (Fig.5.2). This then necessarily facilitates the user-computer interaction and thus raises the system intelligence.
Such a representation must also be explicit in the sense that it is neither just a set of kinematic parameters and spatial positions of each body which are derived from the information at the embodiment level nor based on a particular view of mechanisms for analysis of kinematic motion behaviours of mechanisms. CAD systems in the first place usually have resulted that the human designer is forced to begin his design at the embodiment stage for conceptual design activities. For example the specification of the so-called mating condition (Fig.5.3) is usually required to infer the kinematic relationship between two bodies (Kim & Lee, 1989; Oliver & Harangozo, 1991). This at least leads that the human designer is confronted with unnecessary details for the conceptual design (an example will be provided in the next section). CAD systems in the second place have resulted that the human designer is forced
to learn underlying theories founding those particular views and involved in the too detailed level typically needed for input specifications of those systems. (A more detailed discussion of related work regarding both aspects can be found in Section 5.8.)

![Diagrams](image)

**Legend:**  
- Ground point; L1, L2, and L3 will not influence the kinematics analysis

**Figure 5.4. Two Components Form a Sliding Motion**

Moreover on certain occasions the absence of GMM could cause the failure of a design. A typical example in the conceptual mechanism design is shown in Fig.5.4. For the purpose of kinematics, it is only necessary to specify that body i and body j are relatively sliding. The precise value of the length of the concerned bodies is not important. However, in the computer system which initiates the design at the embodiment design stage and generates the kinematic behaviours of mechanisms based on the mating condition, the human designer is inevitably forced to specify such length precisely. A dilemma arises in that on one hand the precise value of such length must depend on the relative sliding range of body j (with respect to body i)—a result of the kinematic analysis, while on the other hand the kinematic analysis based on the mating condition of the assembly modelling depends on this precise value (notice, in Fig.5.4c, the situation of no mating when such length is too short).

Fig.5.5 shows a comparison of two situations, (a) without GMM and (b) with GMM, with respect to the kinematic and dynamic analysis of the mechanism, where the user activity (u), the computer activity (c) and the user-computer interaction activity (c/u or u/c) are displayed.

**Situation (a):** Suppose that the analysis package based on the finite element approach is used, such as SPACAR (Van der Werff, 1983a). Without GMM, the end-user has first to make a finite element model of mechanism structure A-B-C on the most left of the figure (user activity: u1), then label the elements as well as the nodes (user activity: u2) and finally write the input file specific to the package (user-computer interaction: (u/c)1). The package will calculate the results which are presented as node coordinates and element deformations, etc.
(computer-user interaction: (c/u)2). The end-user should further interpret those finite-element-relevant results into the representation relevant to the mechanism object (user activity: u3).

**Figure 5.5. GMM versus Design Automation and User-friendliness**

**Situation (b):** The end-user merely tells the computer his mechanism structures with his request on the view of behaviours of certain mechanism objects, e.g. the rotation of body A-B (user-computer interaction: (u/c)1). The information about the structure of his mechanism with his request is stored in GMM. The computer program interprets GMM into the input file of the analysis package (computer activity: c). The results generated by the analysis package are further interpreted by the computer program with the help of GMM into the representation relevant to the mechanism object (computer activity: (c/u)2). The above comparison shows that both design automation and user-friendliness of the system will be increased with the presence of GMM. The discussion also implies that a particular view (e.g. the finite element view of mechanisms) for exploring behaviours of the mechanism can automatically be derived (Details will be presented in Chapter 8).
5.3 Requirement on and Philosophy behind Developing GMM

In Section 5.2 the need of GMM has been elaborated with respect to the intelligent computer-support for the conceptual machine design. More requirements will be proposed on developing GMM to further raise the intelligence of CIMOME based on GMM. Basic ideas behind developing GMM are described with the objective to meet those requirements.

The *Intelligence* of a system is an inherent property which is externalized during the user-computer interaction. The system intelligence can then be built in developing a system. Three aspects are usually considered for raising the system intelligence, i.e. design automation, user-friendliness and data access speed. These aspects are all relevant with an application object model in a system (i.e. GMM in the present study), e.g. the design automation will be essentially improved if in the computer there is an explicit representation of application objects (see Fig.5.5 and Fig.1.5), while more recent observations (Van Ballegooijen, 1993; Zhang & Van Dijk, 1994b) point out the importance of an application object model to the increase of the user-friendliness of an interactive system.

*User-friendliness* can further be enhanced during developing an application object model with consideration of (a) how to allow for different views of the same object, (b) how to allow human designers (end-users) to use their domain-specific terminology to interact with the computer and (c) how to allow for raw data existing in the computer system especially at the conceptual design stage.

![Diagram](image)

*Figure 5.6. Two Equivalent Representations of the Frame Element*

An object (physical or conceptual) may be perceived or viewed differently in different users' contexts. Fig.5.6 shows a case in the mechanism design where the mechanism frame is viewed either as two ground nodes or as a static bar; the difference could be because of different customs of the end-user or at different design stages. It is better to represent both views and specify their equivalence with respect to the semantics of the mechanism frame.
concept in GMM in order that the system is of better user-friendliness\(^2\). To support different views of the same object the key is to specify various dependencies among different representations in an application object model.

The data at a certain design stage that needs to be refined or probably neglected at later design stages is called raw data. The raw data is strongly related to the conceptual design stage because on one hand the information at the conceptual design stage concerns design solution principles which are essentially non-geometric, while on the other hand the representation of those principles may need geometrical means. For example to represent a type of mechanism as a potential solution to a mechanism design problem a human designer may draw the mechanism sketch (such as the mechanism shown in Fig.5.1b) on the screen through a graphic user interface; at this stage precise values of the coordinates of nodes A0 and B0 are not necessary. At a later design stage, e.g. the kinematic analysis, the values of the coordinates of nodes A0 and B0 become significant and thus must be refined (or given precisely). The case depicted in Fig.5.4 is another example. The length of two bar bodies won't influence kinematic behaviours and can be left as raw data at the conceptual design stage provided that the underlying application object model allows for it.

The key points to support raw data handling in an application object model are (a) a place for storing raw data, (b) a means to mark certain data as raw and (c) a means to convert the raw data into the refined data.

End-users can use their domain-specific terminologies provided that an application object model explicitly represents them into the model. In the mechanism design this means that GMM must be based on the knowledge which is only relevant to the conceptual structure of mechanisms instead of any method or theory on which some formulations for the constrained equations of kinematic motion is based. This is also supportive to having an explicit representation of the conceptual structure of mechanisms instead of an implicit, derivative and temporary representation. The end-user's knowledge assumed in GMM is at the undergraduate student's level. (This assumption is part of the end-user's mental model about the designing of the conceptual mechanism; details are presented in Chapter 7.)

Data access speed means the response speed of the computer during user-computer interaction. This will usually be improved if the underlying model is of semantics and views of the designed object are explicitly represented in the computer. Some experiments have been reported on low data access speed when the underlying model is conventional relational (Kaper, 1987; Dürr et al., 1989). Furthermore such a representation must also be persistent in the sense that after a computer-supported design system is terminated, the information is

\(^2\) In (Cutkosky et al., 1992) multiple representations of product was much emphasized to the concurrent engineering.
kept as part of a product model in the secondary memory of the computer. This is because in the human-designer's world this block of information is usually recorded in design documents and it is persistent (in archives).

The discussion above leads to several ideas behind developing GMM. Firstly, possible different graphic representations of mechanism objects are not taken into account in GMM. However, a user-interface must always be built upon GMM, which means that presentations of mechanism objects are inevitably considered. The idea is to separate the presentation model from GMM (more details can be found in Chapter 7). Secondly, an object-oriented philosophy will be adopted. This is geared with the idea that GMM should concentrate on the representation of the conceptual structure of mechanisms, instead of any views needed for any formalism of the kinematic analysis (more details are provided in the next section). Roughly, GMM concerns a snapshot of a mechanism at one position, which is depicted with the kinematic mechanism diagram (Zhang, 1991). Thirdly, using hybrid semantic database modelling approach. This is encouraged by more and more established modelling building blocks for representing interesting application semantics (see Chapter 3). Other advantages of using database modelling approach are to facilitate the persistence of GMM and to promote the information sharing of GMM by as many as different tools to be integrated. Finally, GMM shall be organized as a generic model in the sense that (1) it is merely a template while individual mechanisms are instances of the template, (2) this template can be extended without damaging existing parts, which again calls for using the data abstraction generalization/specialization (one of the fundamentals for both semantic data modelling and object-oriented data modelling methods) and (3) from GMM various views needed for various tools acting on the conceptual design stage can be derived.

5.4 An Object-based Description of Mechanisms

The aim of this section is to make a formal description of the application semantics, i.e. the conceptual mechanism to be represented in the computer.

Traditionally, a mechanism is described as a group of links connected by kinematic pairs (Sandor & Erdman, 1984). A link is an assemblage of a number of parts which share the same motion and is called a rigid motion unit. Fig.5.7 shows several examples of one link. The geometry of a link has to be specified via kinematic pairs of this link, which depends on the number of kinematic pairs and the allocation of these kinematic pairs on an arbitrarily defined local coordinate system attached on this link (ISO-10303 Part 105, 1991; Shah & Rogers, 1993). The essential points of the link concept are that

- link as a geometric entity is not explicitly represented, albeit it is a carrier of motion, and
- types of link do not make sense because of the unpredictable number of kinematic pairs.
As a result a communication gap between design and manufacture is formed (Zhang, 1991; Iwata et al., 1992), that is, an object at the design stage could considerably be different from that at the manufacturing stage. For example at the design stage the link as shown in Fig.5.7a will become one bar and one gear at the manufacturing stage. The information of objects contained in a kinematic mechanism diagram is actually a kind of abstraction from those volumetric objects at the embodiment design stage by neglecting the information, e.g. materials, etc. This information is of structural nature, although it is more abstract. The link concept above does not make sense of this point.

Furthermore, it is observed that links belong to the category of object views of the conceptual structure of mechanisms (see Chapter 3 for the description of object and object view concept). In the first place it is a motion view of mechanisms. In the second place it is a molecular constructive view in the sense that more primitive concepts need to be explored.

Kinematic pairs require two links to have physical contact with a relative motion. If we restrict ourselves to planar mechanisms the kinematic pairs are: rotating, sliding and high pairs. Some meaningful joints such as two wheels connected with a belt and two parts connected with a spring are missing. The kinematic high pair indicates that a motion transmitted between two links is through a geometrical point (in planar mechanisms) contact of the profiles of two links. Therefore the motion transmissions between two gears and between a cam and its follower bar are considered to the same. Although the concept of the kinematic high pair captures the similarity of the above two cases in the form of geometric contact it neglects the significant difference between gear and cam transmissions. The information reflected from the kinematic high pair is on the molecular level in the sense that more primitive concepts about connection of two machine bodies need to be explored.

The discussion above implies some drawbacks of the description of the conceptual structure of mechanisms based on the link and kinematic pair concepts with respect to our needs and requirements. The essential point is that the concept of the link and kinematic pair is not object-oriented (or mechanism structure-oriented). Therefore an alternative description of the
conceptual structure of mechanisms is envisaged, which should be object-oriented or structure-oriented and at a level higher than the embodiment mechanism structure.

![Diagram of a gear at different levels of abstractions]

**Figure 5.8. Gear at Different Levels of Abstractions**

To give an impression of our idea, fig.5.8 shows two abstractions of a gear body, one at the embodiment level and the other at the conceptual level. Generalizing this simple example, a new description of the conceptual mechanism is introduced below.

A conceptual mechanism is constructed with two classes of building blocks: *atomic-body* and *atomic-joint*. Atomic-bodies are those entities that meet the following conditions:

(a) They are well identified in the machine building community.
(b) Their functionality exists in a qualitative sense.
(c) It is impossible to divide them into smaller entities which comply with the requirements (a) and (b) above.
(d) Division does not lead to the same type of atomic bodies.

Examples of atomic-bodies are: *gear, cam, bar, wheel, belt, spring*, etc. In terms of the above definition, the cases shown in Fig.5.7 will be considered as two or three atomic bodies. Atomic-joints are those relationships that meet the following conditions:

(a) They are well identified in the machine building community.
(b) Their functionality exists in a qualitative sense.
(c) There is a direct connection (but not necessary contact) between two respective bodies.

---

3. A conceptual mechanism is, hereafter used, the same as a conceptual mechanism structure.
Examples of atomic-joints are: rotating, sliding, fixed (or rigid), geared, belt-drive, chain-drive, spring, pure-rolling, etc. In terms of the description above, the transmission characteristics of a couple of gears and a cam with a follower bar are distinguished. This is necessary for the system to be of user-orientation since in the human designer’s mind the gear transmission and cam transmission follow different and matured design philosophies.

It is also implied in the description above that a certain entity, e.g. spring, etc., can be viewed as a joint while in another situation it can be viewed as a body. This flexibility provides the designer with an opportunity to view the same object differently with respect to the design contexts (see the discussion on the issue 'different views of the same object' in Section 5.3).

Any mechanism can then be constructed in the following way.

- **Firstly**, a kinematic chain is formulated based on the two sets of building blocks above. A kinematic chain is defined as a set of pair-connections. Each instance of the pair connection is therefore in the form of a triple: <atomic-body i, atomic-body j, a-joint>.
- **Second**, several atomic-bodies and/or elements of atomic-bodies in a kinematic chain are imposed to be static relative to a common reference (e.g. ground). The group of bodies and elements of bodies which are static constitute the so-called mechanism frame body. The kinematic chain with the frame body is called a potential mechanism in the sense that an applicable mechanism is not guaranteed yet.
- **Finally**, one or more prescribed motions (normally called input motions) are imposed on the relative motion between two bodies or between elements (on certain bodies) with another body. Since the frame body belongs to the atomic-bodies, any relative and absolute motions are equally defined. A potential mechanism with the defined input motion is called a mechanism. Any actual mechanism is formed when the requirements of the user on a mechanism (output motion) is determined.

### 5.5 Conceptual Schema of GMM

In this section a conceptual schema of GMM, based on the semantics described in Section 5.4, will be presented using the knowledge about the data modelling introduced in Chapter 3. How the model is in conformity with our many thoughts earlier mentioned, e.g. some aspects for raising the system intelligence and the generic model concept will be demonstrated. The presentation will focus on the illustration of key ideas.

#### 5.5.1 Modelling of mechanism atomic-body

Fig.5.9 shows the model of mechanism atomic-bodies. The common properties of all types of bodies, e.g. mass, motion-status, etc. are gathered to a generic class object 'MECH-
Figure 5.9. Data Representation for Mechanism Atomic Body

BODY. As mentioned earlier, at the conceptual design level the geometric characteristics of mechanism atomic-bodies are those features that will only influence the kinematic motion, e.g., for a bar body two end-nodes belong to those features. The kinematic parameters are determined by these features. The number of these features is not predictable in the sense that for an individual body these features are produced dependently of its connections with other bodies, e.g., a bar shown in Fig. 5.10b has four of such features. It is then difficult to explicitly define these features as attributes of a type of bodies because this will lead to an unpredictable number of attributes. Therefore the inherent features and non-inherent feature concept is introduced.

Inherent features of a body are the necessary characteristic elements that generically distinguish the underlying type of a body from others (Zhang, 1991), while the remainder of features are non-inherent. For example a bar body has the inherent feature: two end-nodes
Chapter 5  
A Database Model for the Conceptual Structure of Mechanism

\[\text{Note: L1,L2 and L3 belong to the kinematic parameter}\]

\[\text{Figure 5.10. Inherent-features versus non-inherent features}\]

\[\text{Figure 5.11. Alternative Data Representation of Non-inherent Feature Nodes}\]

(Fig.5.10a). Node elements P1 and P2 in Fig.5.10b are non-inherent. The inherent feature of a body is independent of connections of the body with others and thus the number of the inherent features of a type of bodies is predictable. Inherent features with relevant kinematic parameters are explicitly defined as attributes of a type of bodies (see Fig.5.9a) while non-inherent features are represented together in a class object called 'NON-INHERENT' (Fig.5.9b) by following the data modelling guideline-1 (Chapter 3). In this class object kinematic parameters are represented by the attributes Rx, Ry, Rθ which describe the relative position of non-inherent features with respect to the reference system on each body\(^4\). An example is shown in Fig.5.9c. One might think of an alternative scheme to define non-inherent features by means of the set-valued attribute notion (see Chapter 3), i.e. defining an attribute 'non-inherent-node' (as a set-valued attribute) to each type of body (Fig.5.11a). The instance of class 'BAR' could be as shown in Fig.5.11b. This scheme has, however, difficulties in representing kinematic parameters related to those non-inherent features explicitly.

\(^4\) The reference system on each body will be illustrated in Section 5.5.3.
Manipulation of a group of bodies as a whole may be needed in a design step. This group of bodies is called *composite body*, which corresponds to the composite object concept in the data modelling (Chapter 3). An example can be illustrated. At the conceptual design stage one gear and one bar as shown in Fig.5.12a are separately processed. Considering the later design stage (*e.g.* embodiment design stage) these two bodies might be considered as a whole (Fig.5.12a, alternative II). The information that the body depicted in alternative II of Fig.5.12a is referenced to two bodies at the conceptual design stage must be specified explicitly (see also Section 5.6 for a more general discussion about the connection between the conceptual and the embodiment stages). This leads to the definition of a class object 'COMPOSITE-BODY' (Fig.5.12b). Remarks concerning the definition of this class object are given in the following.

- A composite body is assigned a unique object id and can have its own attributes such as 'purpose'. An object id will be served as a reference for a composite body to relate to its component bodies, see the example shown in Fig.5.12c.
- A composite body is also a sub-class of class 'MECH-BODY'. So that it can inherit the attributes of 'MECH-BODY'.
- The value of attribute 'composite-criterion' can either be predefined at the database schema definition level (*e.g.* the criterion that any rigidly joined body is viewed as a composite.
body) or given by the designer through a user-interface. The value of this attribute is
served as a grouping expression of the grouping data abstraction (see Chapter 3).
Eventually both the selection criteria (for composition) and the bodies to be grouped with
the selected criterion can be end-user-controlled.

In the discussion above nodes have been used to describe the geometry of atomic-bodies at
the conceptual design level, see also, from Fig.5.9a, class object 'NODE'. This is in
conformity with our idea a snapshot of the mechanism at one position (depicted with the
kinematic mechanism diagram), mentioned earlier in Section 5.3. As in a two-dimension
sketching, a node plays an important role in defining the geometry of the conceptual body.
An important advantage of doing so is that the system based on GMM can better support the
conceptual design of mechanisms on a sketch-like manner. In such a system, a human
designer describes a mechanism concept in a very similar way as he usually does manually,
i.e. draws a mechanism sketch in a sheet. Another consideration to introduce the node as a
means of defining the geometry of a mechanism is to support different views (or representa-
tions) of the geometry of conceptual mechanisms. A simple example is that the length of a
bar can be specified either by specifying the length of that bar or specifying the coordinates
of two end-nodes of that bar. In the present study, we shall call the first way of specifying
kinematic parameters of a mechanism as explicit representation while the second way as
implicit representation. A kind of external constraints must be specified in the model (as a
kind of rule asserted in an appropriate place of the textual description of the model schema).
For example for the type of body bar, we have:

\textit{GMM External Constraint Rule <1>}: \\
(for class object 'BAR')
the length of a bar = the distance of two end-nodes.

It should be noted that the conversion from the implicit representation of kinematic parameters
to the explicit representation is straightforward while the reverse process itself requires to
solve a set of complex non-linear equations. So in most cases node coordinates in the model
are raw data while the length of a bar is dominant data or refined data. When a designer
starts with only drawing his mechanism concept without explicitly specifying a kinematic
parameter such as the length of a bar body, the length of a bar body might be calculated
directly from the relevant nodes. It is needed to specify the way the length of a bar is given,
i.e. the system might enable to specify the length of a bar as raw data when the length is
calculated only by non-accurate node coordinates. In GMM such a requirement has been
realized at the implementation level by setting a flag variable for each value in the database
as FALSE (raw data) or TRUE (refined data). This facility is directly provided with the used
DBMS called ABC system which has been outlined in Chapter 3. The discussion is equally
applied to all types of kinematic parameters. Some other considerations of using nodes
explicitly will be discussed later.
In GMM the frame body is modelled using *grouping* data abstraction as shown in Fig.5.13. To support different views of the same object, *e.g.* the mechanism frame could be either two ground points or a ground bar body (see Fig.5.6, earlier mentioned in Section 5.3) the multi-grouping mechanism is applied, *i.e.* grouping from class object 'MECH-BODY' and class 'NODE'. The consistency of the information, *e.g.* a static bar with two ground nodes, is equally maintained by GMM-Constraint Rule <1>. Using grouping data abstraction is suitable because the frame body is formed during specifying a mechanism sketch, which is a dynamic process. The frame body is a composite object and may have its own attributes (Fig.5.13). At later design stages a complete frame body may be designed for manufacturing by an appropriate method such as *casting*.

![Figure 5.13. Data Representation for the Frame Body](image)

**Example**

instances in class 'FRAME' for the mechanism described in Fig.5.6a

<#1,'node',A>

<#1,'node',B>

where #1=the object id for the frame body.

**Note:**

predicate-1: motion-status='static'
predicate-2: motion-status='static'

![Figure 5.14. Adding New Body Types](image)
The model shown in Fig. 5.9 is in conformity with the third characteristic of the generic model concept (see Section 5.3). For example in Fig.5.14 it is shown that adding two new types of bodies does not affect the existing model at all. The two types of bodies are placed at different levels of the generalization hierarchy, respectively.

5.5.2 Modelling of the connections

Although the nature of the connection of bodies is a network, it can be viewed as a set of connections between two bodies, the pair-connection. A class object 'CONNECTION' is defined (Fig.5.15a). Several notes regarding this class object are as follows:

- Since the instances of connections: <body-i, body-j, ...> and <body-j, body-i, ...> mean the similar semantics, an external constraint must be set up below:

  GMM External Constraint Rule <2>:
  (for class object 'CONNECTION').
  <body-i, body-j, ...> = <body-j, body-i, ...>.

- The configuration of a mechanism can be determined via two ways. The first is via the nodes in the global coordinate system attached to the frame body. The second is via relative positions between two bodies. As mentioned earlier, the coordinates of the nodes are defined in the global coordinate system of a mechanism and on the other hand the nodes are used in the body description. This implies that the positioning of bodies depends on the nodes. So the first manner has sufficiently been supported. To support also the second manner, a set of attributes which describe the relative position of two connected bodies must be used, i.e. attributes: 'r-x', 'r-y' and 'r-θ' (Fig.5.15a). Also it is important to specify, when dealing with relative angles, the reference body. This is fulfilled with the attribute 'reference-body'. Several examples of specifying values for these attributes considered at the implementation level are shown in Fig.5.15b. It is especially noted that in a certain case those attributes may not explicitly represent the relative position between two bodies, e.g. for a couple of gears connected with a gear joint the signed transmission ratio is used (case (2) of Fig.5.15b) which also represents inner (ratio > 0) or outer (ratio < 0) engaging.

- Two bodies are connected via elements on each body. This is done by selecting a feature node on each body. This leads to defining 'NODE' as an attribute of class 'CONNECTION'. Feature nodes here are served as the pair element from each body concerned. Fig.5.15c shows several examples considered at the implementation level. Case (2) and Case (3) of Fig.5.15c show that feature nodes may not be contacting points. This is an extension to the traditionary pair element concept (Erdman & Sandor, 1984). A feature node is in general a kind of representative of a body to accomplish the total information about a type of connection between two bodies; contact elements are only one of those candidates of body-representatives. When two bodies are joined with a relative translation
Figure 5.15. Data Representation for Connection

it does not make much sense to select any specific contact element on each body as pair element because the contact element is actually a line. The strategy in GMM for this situation is implied in the example given in case (3) of Fig. 5.15c. As mentioned earlier in
Section 5.5.1, the feature node also contributes to the description of the geometry of a conceptual body. In this way the complete information about a connection between two bodies is represented.

There actually exists an alternative way to specify a connection between two bodies without using the feature node concept. For each body feature elements can be defined in such a way that the position of the feature elements is known to the local coordinate system of individual bodies (Fig. 5.16a). The connection information can therefore be specified via feature elements as shown in Fig. 5.16a with an example. This idea is rooted in the assembly modelling culture where connection between two geometric entities is specified via the mating condition (see Fig. 5.3). The feature element here could be thought of as a kind of mating elements. The current version of GMM supports the feature node concept, although the possibility for GMM to support both manners could be investigated.

![Diagram showing warehouse and assembly design site](image)

**Figure 5.16. Alternative Data Representation of Connection Semantics**

The key point to have such a support is to model the relationship between feature nodes and feature elements. The example of Fig. 5.16a implies that the functional correspondence (feature node : feature element) is 1:n which means that one feature node corresponds to more than one feature element. Noticing this fact and considering that a feature element is local to an individual body, the class object 'FE-FN' is defined (Fig. 5.16b). Other adaptations include the replacement of the attribute 'NODE' by 'feature-element' in the classes 'CONNECTION' and 'MECH-BODY', respectively.

Explicit use of nodes seems better to support the conceptual design in such a scenario that a design is started with sketching the kinematic mechanism diagram. As such, while a kinematic mechanism diagram is being drawn, corresponding nodes are created in class 'NODE' and written into class 'CONNECTION' as a marker for the connection of the two respective
bodies. Such a marker is further inherited to the description of individual bodies. Node coordinates also provide information of a particular mechanism branch, which can be stated as: given the same kinematic parameters and DOF (Degree of Freedom)-values there may exist more than one configuration (Fig. 5.17).

*Figure 5.17. The Mechanism Branch Information versus Node Coordinates*

The definition of class 'CONNECTION' can easily be extended when needed. For example, to represent the cam mechanism, extra information about the connection must be represented, *e.g.* the contact form between a cam and its follower bar and pressure angle, etc. Fig. 5.18
shows the inclusion of the extra information, by defining a class 'CAM-CONNECTION' as
the sub-class of class 'CONNECTION', without affecting the existing part. The implementa-
tion of a system that uses GMM as its underlying object model for designing complex cam
mechanism has been successfully accomplished (see Chapter 9).

5.5.3 Modelling of input-output motion
Input motions are performed by drivers to enable a potential mechanism as a motion
generator. Output motions are those interesting to an actual purpose of a mechanism in the
designer's mind. Usually the input-output relationship is an important behaviour of a machine
to be explored. To be more general any motion is viewed and defined in the relativity context
in GMM.

![Diagram showing the relationship between two local coordinate systems](image)

**Note:**
- X1-Y1: local coordinate system on body-1.
- X2-Y2: local coordinate system on body-2.
- α12: the relationship between two local coordinate systems.
- θ12: the angle that the end-user wants to view.
- R1,R2: User-oriented reference on each body.

*Figure 5.19. User-oriented View versus System-oriented View*

In a multi-body mechanical system (or mechanism) there is a local coordinate system attached
to each body, e.g. the Denavit-Hartenberg system (1955) and Sheth-Uicker system (1971).
The relative position of body i with respect to body j is completely determined by the
relationship between two corresponding local coordinate systems. Apart from the local
coordinate system, a user-defined reference system (a vector, a point, or a coordinate system)
in introduced because the motion expected for view by the designer may not be necessarily
measured in the local coordinate system. Fig.5.19 shows an example. In this example an end-
user expects to view the displacement θ(12) between two vectors R1 and R2 which are,
respectively, fixed to two local coordinate systems X1-Y1 and X2-Y2. The definition of two
references R1 and R2 is completely user-oriented. In general there are four types of user-
oriented references: a point, a free vector, a fixed vector and a coordinate system. Four cases
of motion in the context of the user-oriented reference system are considered in GMM
(Fig.5.20):

(a) on the reference body: a free vector; on the target body: a free vector.
(b) on the reference body: a fixed vector; on the target body: a point.
(c) on the reference body: a coordinate system; on the target body: a point.
In the following, such references are called *user-oriented reference system*. To raise the system intelligence in the aspect of the user-friendliness, GMM supports these user-oriented reference systems in the sense that they are explicitly represented in GMM, see the later discussion.

In contrast, the local coordinate system is much more *system-oriented* and its definition may be subject to some conventions to facilitate the establishment of kinematic motion constraint equations (Zhang & Klein Breteler, 1990a). The local coordinate system is, however, necessary to unambiguously maintain constraints among bodies of a mechanism. In GMM the idea is to keep the local coordinate system implicitly available in the sense the GMM system\(^5\) will automatically create and maintain it (whenever necessary) in terms of the structure of a mechanism and a set of predefined conventions. Since GMM at present considers only planar mechanisms the local coordinate system is simply represented by a local (fixed) vector. Fig.5.21 shows a part of the conventions for defining local vectors with respect to types of body. The local coordinate system can be created based on this local vector (by taking the

---

\(^5\) GMM system contains (1) GMM, (2) a set of processes to manipulate GMM and (3) a user-interface upon GMM.
vector as the x-axis and rotating 90° from the x-axis in a counter-clockwise to get the y-axis).

![Image](image_url)

**Figure 5.21. Local Coordinate Systems versus Types of Body**

![Diagram](diagram_url)

**Figure 5.22. Data Representation of User-oriented Reference System**

In terms of the discussion above, a class object 'U-REFERENCE' is defined (in Fig.5.22a) to represent user-oriented reference systems as well as their relationships with the system-oriented reference system, i.e. the local coordinate system. Several remarks concerning this definition are given below.

104
The common properties of each type of user-oriented reference system define a generic class 'U-REFERENCE' (U = User). These properties are, e.g. the name that the end-user may want to assign to each reference system and the mechanism body that the reference system is related to.

The value of the attributes 'local-x', 'local-y' and 'local-angle' (defined at individual sub-classes of class 'U-REFERENCE' level) is with respect to the local coordinate system of the body in which the user-oriented reference system lies (Fig.5.22b). Since both nodes (the origin of the local coordinate system and the origin of the user reference system, see Fig.5.22b, O-LCS and O-URS respectively) are defined in the global coordinate system, there exists a data redundancy among these two nodes and the local coordinate attributes. To support both views of specifying the origin of a user-oriented reference system (Fig.5.22c and Fig.5.22d), an external constraint is therefore defined below:

**GMM External Constraint Rule <3>:**
(cross-reference among sub-classes of classes 'U-REFERENCE' and class 'MECH-BODY', respectively)

\[ \mathbf{O}_{URS} - \mathbf{O}_{LCS} = [\text{local-x, local-y}]^T \] (where \( \mathbf{O}_{URS} (\mathbf{O}_{LCS}) \) is a column matrix)

When the end-user does not establish his own reference system, the GMM system will take the system-oriented reference system as default. In this case the value of the attribute 'ref-type' will be 'system-point' or 'system-vector', etc. and the precise information about the reference can be found in the local coordinate system of the corresponding body.

The discussion above implies that various types of relative motions to be viewed by the end-user can be represented by relating two user-oriented reference systems. This leads to the definition of the class object 'IO-MOTION' (Fig.5.23a). More detailed information about input and output motions is represented in Fig.5.23b. In Fig.5.23b the attribute 'motion-type' is implied or derived from the combination of two attributes 'ref-reference' and 'target-reference' of its super-class 'IO-MOTION' and expresses all the four cases as shown in Fig.5.20, e.g. path, rotation, etc. The attribute 'motion-order' means the required order of motion, for example, zero-order means the displacement. The attribute 'number-of-point' means the total number of input motion steps for which the output motion is to be calculated. Fig.5.23b also represents for each type of motion the motion primitive element, for example, for a path two motion quantities (x and y) are needed.

Input motion differs from output motion in that the input motion value is predefined while the output motion value is calculated by analysis systems. For certain types of input motions their values can be specified by a mathematic formula with certain parameters given, instead of a set of discrete points. This leads to the definition of class object 'INPUT' which is a sub-class of 'IO-MOTION'. For the input type such as 'constant-velocity (equal-step)' the formula would be: \( S = Vt + S0 \) (where, \( S \) = displacement, \( V \) = velocity). When the end-displacement (Se)
is given, the displacement values: S1, S2, ..., Se (for t=t1, t2, t3, ..., te) are calculated.

Figure 5.23. Data Representation of Input/output Motions

Figure 5.24. Data Representation of Transfer Function with Respect to Structure
The behaviour function (its definition can be found in Chapter 2), or termed transfer function in the mechanism (Van der Werff, 1977), can be represented as shown in Fig.5.24 by putting two relevant variables together. It is noted that here the class TRANSFER-FUNCTION has contributed to the representation of a behaviour function in the aspect of the physical meaning of variables of a function. The form of a behaviour function has already been represented with the Generic Function Model (GFM) in Chapter 4. So a clear picture of the relationship between GMM and GFM can now be depicted as shown in Fig.5.25.

![Diagram](image)

*Figure 5.25. Integration of GMM and GFM upon Transfer Function*

### 5.5.4 Modelling of an actual mechanism

An actual mechanism consists of a group of connected bodies with defined input and output motions. In the viewpoint of the data modelling an actual mechanism is simply a composite object which consists of all these objects defined above. Using the composite object constructor developed in Chapter 3, Fig.5.26 defines the data representation for an actual mechanism. The mechanism object as a composite object can have its own attributes (Fig.5.26).

![Diagram](image)

*Figure 5.26. Data Representation of an Actual Mechanism*
5.6 The Link between GMM and the Machine Assembly Model

Assume there are two design sites, one for the conceptual design and the other for the embodiment design (Fig. 5.27). The link between these two sites must be available to achieve a consistent design. Since the information of the conceptual structure of mechanisms and the embodiment structures mechanism is, respectively, stored in GMM and an assembly model, such a link is responsible for maintaining the change propagation between GMM and an assembly model. The link actually contains a set of processes invocable. The establishment of those processes depends on the two models concerned. For example when an attribute in GMM also appears in an assembly model with the same semantics, the process to link these two attributes is simply a unit mapping.

![Diagram showing the link between GMM and the assembly model.]

Legend: C-site = Conceptual design site; E-site = Embodiment design site;  → link in the human user's world;  → link in the computer world

Figure 5.27. A Scenario for two Design Sites: Conceptual and Embodiment

There have been many studies on the assembly modelling (Sodhi & Turner, 1994; Shah & Rogers, 1994; Oliver & Harangozo, 1991; Srikanth & Turner, 1990; Kim & Lee, 1989; Lee & Gossard, 1985; Kitajima & Yoshikawa, 1984; Tilove, 1983). In the present study, only key elements of the assembly modelling are considered.

In any assembly model the generic issues are the modelling of components and the modelling of relationships among components. For the component modelling there are basically two trends: one directly based on the solid modeller (Lee & Gossard, 1985) and the other based on the feature concept (Shah, 1991). For the relationship modelling there are also basically
two trends: one based on the explicit representation of relationships between two contact bodies in the transformation matrix (Tilove, 1983; Shah & Rogers, 1994) and the other is based on the mating condition (Ko & Lee, 1987). In the latter case the part position is derived in the site of the end-product.

For not being restricted with one particular assembly model, we assume that the assembly model considered for the present discussion is feature-based (in component modelling) and has a local coordinate system (on component) which might be used either for the explicit representation of a transformation matrix or for mating conditions.

For the simplicity of the later discussion, the link is decomposed into two links: (1) L(C->E): from the conceptual to the embodiment and (2) L(E->C): from the embodiment to the conceptual. A separate discussion of L(C->E) and L(E->C) is briefly given below.

5.6.1 The link L(C->E)
Consider a bar body, as an example, at two design sites in the context of planar motions (Fig.5.28). Some requirements and feasibilities in establishing this link can be studied. The problem that this link should tackle can be stated as:

*Whenever the kinematic parameter at the conceptual design site is changed, what and how adaption on the component at the embodiment design site has to be done.*

![Figure 5.28. A Bar at Two Design Sites: Conceptual and Embodiment](image)

As implied above, the first thing is what, i.e. what parts of a total component at the embodiment design site are affected. This requires a scheme to specify the correspondence between the length (kinematic parameter) and those parts needed for adaption. Depending on individual assembly models, several possible correspondences might exist:

**Case (1)**
When in the assembly model a kinematic parameter is just a feature parameter of the assembly model, the correspondence is simply, as such:

\[ \text{the kinematic parameter (kp\#1) of a bar body (b\#2) = the feature parameter (fp\#4)} \]
where, 

\#i = the number index (in the sense of database, it is an object id),
kp = Kinematic Parameter, b = Body, and fp = Feature Parameter.

Case (2)
When in the assembly model such a kinematic parameter is only implied in the location of two corresponding holes (the location can be specified in terms of the component local coordinate system, xE-yE-zE, see Fig.5.28b), the correspondence might be specified as:

the kinematic parameter (kp\#1) = relative distance between hole (h\#1) and hole (h\#2)

where, h = Hole.

The semantics above must be represented in a database model (hereafter C-E link model for short). This is possible after that both GMM and an assembly model have been instantiated or created. These semantics can better be represented in a C-E link model simultaneously with the process of embodying a conceptual mechanism.

The next issue is to manipulate the change of the parts (in an assembly model) in accordance with the change of the kinematic parameter suggested at the conceptual design site. In other words, how the corresponding volumetric objects are changed given a kind of input, say one of its feature parameters (in the case of Fig.5.28c, feature parameter \( \ell \)). The change operation now is quite dependent of how the volumetric objects are specified and created. Assume that in the C-E link model, the case (2) is recorded. Several possible change operations can be described. When parameters a and b (Fig.5.28c) are independent of \( \ell \), the change operation would be: move hole B along xE positive, keep a and b unchanged and change parameter L accordingly. When parameters a and b (Fig.5.28c) are dependent of \( \ell \) (say, a = b = 0.25\( \ell \)), the change operation would be: move both hole A and hole B along xE direction so that the location of hole A on the xE dimension is equal to 0.25\( \ell \), etc. The discussion also implies that, in the context of the embodiment stage, parameters to be changed are divided into two sets: independent parameters (e.g. \( \ell \)) and dependent parameters (e.g. a,b,L), see Fig.5.28c. Fig.5.29 concludes with a general working flow for the link L(C -> E). The work by Schickendatz (1989) actually always assumed that the kinematic parameter of a conceptual body at the conceptual design is always a feature parameter of the corresponding volumetric component at the embodiment design. This idea has been further elaborated by Zhang (1991) such that the embodiment mechanism structure is simply to add some extra attributes (based on kinematic parameters) necessary to describe a volumetric object. While the approach appears to be over simplified, it initiates a more general idea to utilize the modern feature concept. One can view a kinematic parameter as one of the features representing mechanical assemblies at the embodiment design stage. The design process proceeds as such, at the conceptual design stage, kinematic parameters (or features) are
determined, while at the embodiment design stage more features are determined which finally derive a volumetric mechanical assembly. More detailed discussion of this idea is beyond the scope of this dissertation.

5.6.2 The link L(E->C)
Two separate issues are regarded to this link: (1) when the design process starts with the embodiment design, can GMM be derived from an assembly model? and (2) at the presence of GMM and an assembly model, when any feature parameter is changed in the assembly model, what and how is this change propagated to GMM? Inherently the second issue depends on the first issue. Furthermore the process attached on link L(E->C) can mostly be automated. This is because this process can be viewed as simulating the procedure of drawing a kinematic mechanism diagram from a mechanical assembly (see Fig.5.1), for which there is a fixed set of rules available from the domain of design theory of mechanisms.

The work by Kim & Lee (1989) can be viewed as an experiment relevant with the first issue. They demonstrated the possibility to derive, from the assembly model based on the mating condition, the joint coordinate system (or C-LCS in our terms), e.g. Denavit & Hartenberg convention (1955) which is widely used in the analysis of mechanisms including robots. Their work is supportive to the feasibility of the idea, i.e. automating the process from the assembly model to GMM. This is because most of the general-purpose analysis systems are essentially based on the conceptual mechanism structure information. The process from the assembly model to GMM is therefore less complex than that from the assembly model to general-
5.7 Implementation and Preliminary Verification of GMM in Use

The conceptual schema of GMM has been mapped to the schema of the ABC system (see Chapter 3, a dedicated main memory DBMS). GMM can also be written in EXPRESS (ISO-TC184/SC4/WG5, 1991) to promote the exportability to other CAD systems. GMM has been used in many implementations concerning CIMOME. An intelligent editor system (EDMECH, see Chapter 2 and Chapter 7) to support a sketch-like manner of specifying conceptual mechanism structures has been implemented with GMM being the underlying model (Van Ballegooijen, 1993; Stokkermans, 1994; Zhang & Van Dijk, 1994b). The user-friendliness and high data access speed of this system provide a strong implication of the GMM’s features: its semantic naturalness to the human designer and its apparent data structure due to the use of semantic database modelling and the object-orientation philosophy.

The information represented in GMM is complete in the sense that it is sufficient for our purpose to support various design tasks at the conceptual level. In Chapter 8 it will be shown that from GMM a finite element view of a mechanism for analysis and synthesis can be automatically generated via a system called GMMFEM (see also Chapter 2 for the definition of this tool). Those very complex analysis packages are then transparent to the end-user.

Van Veen (1994) implemented a system called CAMDES (see also Chapter 2 for the definition of this tool) for the compound cam mechanism design with the objective to integrate EDMECH, TADSOE and RUNMEC (see also Chapter 2 for definition of these tools). The underlying object model is GMM. With the help of the GMMFEM system, the design automation and user-friendliness reach yet the highest level for the compound cam design in the mechanism design community. Some key points of the implementation of this system will be illustrated in Chapter 9.

Van Wagensveld (1994) demonstrated the possibility from GMM to extract kinematic parameters into her design model for the dimension synthesis (of mechanisms) via optimization. The work is promising in the automatic generation of a finite-element-based optimization model for the kinematic synthesis via a package called OPLAM (see also Chapter 2 for the definition of this tool). Some key points of the implementation of this system will be illustrated in Chapter 9.

From GMM various views for some conceptual design activities (e.g. mobility analysis and type synthesis, etc.) can easily be derived. The information about a kinematic chain can be obtained by suppressing the frame body instance in the data model and grouping any bodies with the fixed joint as one, i.e. the link. The kinematic chain information is needed for the
mechanism type synthesis. Jongsma (1992) has shown the possibility to derive an \textit{adjacency matrix representation} of the mechanism topology from GMM, which is an important tool for mechanism type synthesis. The information about a potential mechanism can be obtained by simply skipping over the class object 'IO-MOTION. This information is needed for the mechanism mobility analysis.

5.8 Discussion and Conclusions

In this section GMM will be extensively compared with related work to predicate the perspective of GMM. GMM is one of the few systems to explicitly represent the information about conceptual mechanism structures. In spite of this, the scope of the comparison will be extended to some assembly model systems because the information of the conceptual mechanism may be implicitly contained in those assembly models. However, only those aspects relevant to the conceptual design support those systems provide are considered for evaluation. Among the total conceptual activities, the emphasis is placed on the kinematic and dynamic analysis, since most of assembly model systems were studied to support this activity from their assembly models. To facilitate the following discussion the research aspects (or dimensions) on the interested issues are set up first. Both related work and GMM can then be discussed in the context of the research dimensions.

5.8.1 Research dimension

Three research dimensions are set up, which are: (d1) philosophy, (d2) component representation and (d3) relationship (among components) representation. By \textit{philosophy} is meant the basic idea behind developing the data representation of mechanical assemblies. Component and relationship among component representations are two generic things that any model must concern. On each dimension we also define 'milestones' in terms of specific viewpoints which will be mentioned below.

\textit{On the dimension of the philosophy} (1st dimension with four milestones).

d1(1): The conceptual design is started at the embodiment design stage. There is no explicit representation of the information about machine conceptual structures. Usually the geometry of the conceptual mechanism structure implicitly resides in volumetric objects. The design starts with the specification of volumetric objects and then relationship between them.

d1(2): The representation of the information about the conceptual mechanism structure is coupled with those particular views for the kinematic and dynamic analysis.

d1(3): The information about the conceptual mechanism structure is recognized with emphasis on motion transmission. The representation is therefore based on the concept \textit{link}--one motion unit.
d1(4): The information about the conceptual mechanism structure is recognized as a higher level of abstraction from the embodiment mechanism structure. The starting point of the data representation is somewhat based on the idea: a snapshot of the mechanism at one position (depicted with the kinematic mechanism diagram).

![Diagram](image)

(a) The research dimension on Philosophy

(b) The research dimension on component representation

(c) The research dimension on relationship representation

*Figure 5.30. Research Dimensions on the Data Representation of Mechanism Structures*

The milestones on this dimension are set in terms of the support capability for the conceptual design (see Fig.5.30a).

*On the dimension of the component representation* (2nd dimension with three milestones).

d2(1): Utilizing data structures in general-purpose solid modeller systems, in most cases the boundary representation (Tilove, 1983).

d2(2): Utilizing database modelling approach to establish object models which include not only volumetric information but also some information such as material properties. Such models are, on one hand, communicated with general-purpose solid modeller systems via such as IGES, on the other hand, manipulated by customized assembly analysis systems (Martens, 1992).

d2(3): Feature-based representation. The solid modelling is transparent to the end-user. For example to specify a slot on a shaft the end-user may only need to tell the system that he needs a slot on a certain place on the shaft. The system will automatically formulate a solid model in terms of specified features (Shah, 1991).

The milestones on this dimension are set in terms of the support capability for a user-oriented man-computer interface (see Fig.5.30b). The user-orientation requires a system to allow the human designer to use their domain-relevant terminology and knowledge at best.
On the dimension of the relationship representation (3rd dimension with four milestones).

d3(1): Specification of the transformation matrix between relevant local coordinate systems on bodies. Early work (Tilove, 1983) requires the designer to specify the local coordinate system and manipulate variables such as the rotational displacement on a revolute joint. The system can subsequently assemble a group of volumetric objects using the information from the transformation matrix, the local coordinate system and the geometry of volumetric objects. When the idea is applied to the mechanism containing closed loops, the dependency of those joint variables can be formulated by establishing the closed loop equation, see Kitajima & Yoshikawa (1984).

d3(2): Explicit specification of mating conditions. This way directly asks the end-user to work on the solid modelling level.

d3(3): Implicit specification of mating conditions. This way improves the d3(2) above by using a vocabulary which is closer to the end-user’s world, e.g. constraint and attachment, etc. The mating conditions such as two face contact and two axial co-line are automatically derived.

d3(4): Using relationship features such as pin joint (Sodhi & Turner, 1994).

The milestones of this dimension are set in terms of the same viewpoint as the 2nd dimension.

5.8.2 Related work

The work performed by Tilove (1983) falls into the research dimension <1,1,1>. The main idea of his work is to establish local coordinate systems on each volumetric body and therefore the relationship among a set of volumetric components is replaced with the relationship among local coordinate systems. The system requires the end-user to specify both local coordinate systems and variables which constitute the relationship between two local coordinate systems. The work was only focused on the open-loop mechanisms.

The work reported by Kitajima & Yoshikawa (1984) falls into <1,1,1>. The main idea of the work is to view the construction of a mechanical system as: (1) several volumetric objects form a part, (2) the geometry of a part is implied in the geometry of volumetric objects and their static relationships (such as the face to face contact), (3) a group of parts forms a more complex object via a fixed connection among these parts, and (4) between two complex objects there is relative motion. All the relationship between objects at different levels is represented by the transformation matrix related to two concerned objects. The system requires the end-user to construct mechanical components from the volumetric object level. For the kinematic analysis the system automatically detects closed loops and then formulates the constraint equations of kinematic motion. This is seen as an major advance over the work by Tilove (1983). The potential problems are identified below:

- The end-user must designate, among a set of parameters, variables or kinematic
parameters. This implies that the end-user has burden to specify local coordinate systems on both the part level and the complex level because they are related to the definition of variables and kinematic parameters.

- The end-user has to be involved in unnecessary work for the conceptual design (e.g. the kinematic analysis and synthesis), such as the definition of volumetric parts and the local coordinate system of parts. This problem is inherent to all those systems that start with modelling at the 3D volumetric object level.

- It is always required by the system that the number of unknowns should be equal to the number of equations. Over-constrained mechanisms are therefore not covered.

- The relationship between components only relies on the transformation matrix, in which some kinematic relationships can not be represented, e.g. the transmission ratio of a pair of gears. The kinematic analysis of the mechanism containing gears is therefore unsolvable with their system.

- The kinematic parameters are implicit in those volumetric objects and their relationships. The end-user needs to relate those implicit values regarding volumetric objects to kinematic parameters which are explicit in his mind.

The work by Kimura et al. (1984) is different from (Kitajima & Yoshikawa, 1984) in the sense that Kimura et al. (1984) started to separate the conceptual structure from the embodiment structure. The idea was only very globally described. There was no description of body types at the conceptual mechanism level and the relationship among conceptual objects relies on the transformation matrix related to concerned objects and some expressions for those relationships which are difficult to be represented in the transformation matrix, e.g. the transmission ratio of a pair of gears. The author recognized that further efforts were needed on a more formal framework of the approach to describing mechanical systems and it was difficult for the end-user to create complex internal relations. The work generally falls into the research dimension \( <3,1,1> \).

Leigh et al. (1989) introduced a concept called model space to represent individual components. The model space was in principle similar with the complex unit defined by Kitajima & Yoshikawa (1984). The relationship between model spaces was represented by the transformation matrix which follows the Denavit-Harterberg notation (1955). The work differs from (Kitajima & Yoshikawa, 1984) in the sense that the information of the loop which constrains joint variables must be provided by the end-user. To use the system the end-user must define local coordinate systems on each model space. Their work generally falls into the research dimension \( <3,1,1> \).

The work by Rocheleau and Lee (1987) falls into the research dimension \( <1,1,2> \). The data structure to represent mechanical assemblies is Lee-Gossard's well-known approach 'virtual link' and mating conditions (Lee & Gossard, 1985a). The method advanced the assembly modelling in two aspects: (1) interactive specification of mating conditions and (2) using three
rotational and three translational quantities to replace twelve elements in a transformation matrix. Thus the order of a set of non-linear equations resulting from the mating conditions is reduced. They recommended the work for generating the input file for the ADAMS analysis package. The inherent problem to use mating conditions to infer the component position is the difficulty in solving a set of larger complex non-linear equations. For example for a simple pivoted four bar linkage, 39 equations with 18 unknowns must be solved (Oliver & Harangozo, 1992).

Kaper (1987) and Langbroek (1989) respectively developed the database models for the conceptual mechanism. They first view a mechanism as a set of finite elements and the model was based on this view. This view is fundamental to the analysis packages such as SPACAR (Van der Werff, 1983a). So the work falls into the 1st research dimension, in milestone nr.2.

There are some commercial mechanism analysis packages such as APPLIED MOTION. Data structures to model the conceptual mechanism are seldom reported. In terms of some external behaviours of those systems, one might get some implications about their underlying data structures. For example APPLIED MOTION only allows for the frame definition via ground points, which implies that a fixed gear on the ground can not be viewed as a part of the frame. Its functionality in specification of a designed mechanism is therefore limited. Another example is that APPLIED MOTION explicitly represents a joint as a point on the screen for the end-user to access. When there are more than two bodies joined with the same point, ambiguous interactions might happen (Fig.5.31a). This is because only pointing to a point does not sufficiently specify two concerned bodies among a group of bodies (more than two). The idea to express a joint as a point will also have difficulties in handling the combination of a revolute and a sliding joint (Fig.5.31b).

![Figure 5.31. Possible Ambiguous Meanings in Defining Joint only via Point](https://example.com/figure5.31.png)

There is a research track to exploit the bond-graph as a means to represent structures of artifacts as well as functions of artifacts, more recently see (Malmqvist, 1994; Bracewell & Sharpe, 1994; de Vries et al., 1994). The bond graph-based representation is in its very
nature to represent system function flows, such as energy transformation. The topology and geometry of the artifact is only implicitly represented. Usually the artifact is parametric with the procedure to take care of its graphic display. Since the structure of artifacts is not represented as data structures explicitly in the computer, such systems do not generally allow the end-user directly to add extra physical components on the artifact which is currently being displayed on the screen; instead, when such requirements are needed systems may allow the end-user to operate on the bond-graph-based data structures. Obviously the work on this track belongs to the 1st research dimension, in milestone nr.2. It has been shown by (Malmqvist, 1994) that an important advantage of using bond-graphs is to represent both functions and structures of artifacts in one formalism. Since at each level of the representation of artifact structures there is always a respective bond-graph, the total system behaviours can therefore easily be analyzed as it is generally possible to deduce the state equations from the bond graph. Malmqvist (1994) further concluded that the extension to representing kinematic behaviours with the bond graph needs to be studied.

In 1991 an international working group reported a model (ISO 10303 Part 105, 1991) of the conceptual mechanism structure. This model is based on the concept link—one motion unit (see the discussion in Section 5.4). This further yields that the complex theory about local coordinate systems has to be used to describe the link geometry. However, this model is closer to GMM than all others. Further discussion of the difference between GMM and the ISO model is provided in Section 5.8.3.

5.8.3 Salient points of GMM
GMM recognizes that above the level of embodiment mechanism structures (based on volumetric objects) there is a level called the conceptual mechanism structure. The information at this level should be explicitly and separately represented. This idea is similar with the ISO model. The communication between the conceptual design and the embodiment design can be facilitated by the C-E link model (as described earlier in Section 5.6) making GMM and the assembly model 'two-way' related. This founds the future system that can propagate changes from and to both 'sites': the conceptual design site and the embodiment design site. The relationship between GMM and assembly models is first clarified in the present study. From the viewpoint of integrating design tools (e.g. analysis, synthesis, etc.) Fig.5.32 further exhibits the difference of two patterns: (a) both GMM and an assembly model exist and (b) only an assembly model exists. The implementation following pattern (b) is not only difficult but also very costly as it neglects the common view (i.e. the conceptual structure) of those tools at the conceptual level. In pattern (a) GMM represents the common view of those tools so that their communication with an assembly model is actually divided into two steps: first with GMM and then GMM with an assembly model. It is clear that

---

6 It is noted that the relationship between the ISO model and the assembly model was questioned and left open by the ISO model developing group, see (ISO-10303 Part 105, 1991, pp.97).
GMM plays a double role: both in the horizontal integration and in the vertical integration (see Fig. 1.3 in Chapter 1).

![Diagram](image)

*Figure 5.32. The Role of GMM in Integration of Conceptual Design Tools*

As known, in the machine design the kinematic mechanism diagram represents kinematically-relevant information of the conceptual mechanism structure and belongs to the category of the *engineering drawing*. GMM's philosophy is that the data representation of artifact structures at those known abstraction levels should be of engineering-drawing orientation in the sense that the consequence of the data representation is roughly to make the computer system understand the engineering drawing and some attached documentary information. This leads that GMM is somewhat a representation of a snapshot of the mechanism at one position (depicted with the kinematic mechanism diagram) in the computer. In this aspect, GMM differs from all others.

In developing GMM we imagine that in front of us there is a real assembled mechanism and we are obliged to give the answer to questions such as what a mechanism is composed of. A generalized object orientation philosophy is adopted such that we concentrate on a constructive view of the conceptual mechanism. As a result the traditional *link* concept in the description of the kinematic mechanism structure is thought to be inappropriate because it is a motion-oriented view. The motion-oriented view is attributed to the situation where the geometry of the link must rely on the kinematic pair (the number of pairs and the geometric distribution of those pairs on the link). Whereas in GMM the generalized joint (corresponding to the kinematic pair) is only meant as a kind of relationship between two bodies and the joint does not contribute to the geometry of a body at all. As such, in GMM the local coordinate system is only implicitly represented (at least in planar mechanisms). This necessarily allows the system based on GMM to support the design process in a sketch-like (or engineering drawing) manner.

In developing GMM we have been looking for a unbiased representation. Both the bond graph
based and the finite element based representations are argued as carrying some views needed for the analysis of behaviours of artifacts. The system directly based on those representations inevitably requires the end-user (designer) to work with those views, albeit those views may have nothing to do with his manual way of specifying the structure of his designed artifact and his purpose of examining behaviours of the artifact. To reach our goal we adopted the database modelling approach as a database is to represent interesting semantics of the real world. It has been shown that the database modelling approach enables to fulfil our requirements. (The idea to use the database modelling approach is similar with the ISO model.)

At the model definition level, GMM has some unique features, which are listed below:

- GMM identifies a so-called user-oriented reference system which raises the user-oriented feature of the system. The input and output motion is defined against the user-oriented reference system.
- In GMM the kinematic pair as well as the kinematic pair element concept is extended to a more general concept of joint representative of the body. The starting point of the joint representative concept can be elaborated. When two parties make a relationship with a certain degree or type, the criterion of selecting the representative of each party is such the representative should enable to completely describe that kind of relationship. Applying this common sense knowledge to the mechanism, we remove the condition, as imposed by the traditionary pair element concept, that the pair element should be the contacting point.
- In GMM the fixed (or rigid) joint is introduced. This is technically a getaway that GMM does not need the local coordinate system for the definition of the body geometry as the ISO model does. For the description of the object as shown in Fig.5.33, GMM's philosophy is that this object consists of three simple bar bodies with rigid joints while ISO model’s philosophy is that this is one link with several pair points. In GMM the geometry of this object is implied in the geometry of the individual bar bodies and the quantitative values of the rigid joint. In the ISO model one has to consider that on each joint there is a joint local coordinate system and then the geometry of this object is implied in the transformation matrix of these joint local coordinate systems (link local coordinate system is one of them in arbitrary choice).
- GMM is an integrated representation of both conceptual mechanism structures and behaviour functions. (Most of the other work merely represents the structure.) This feature is unique in the domain of the data representation for a conceptual machine.
- GMM is established as a database model using the semantic data modelling approach. It is therefore relatively easier with communication to other CAD/CAM systems.

To conclude, GMM falls into the research dimensions <4,3,4>. GMM is promising in that the system based on GMM and the idea about the link between GMM and assembly models is theoretically capable of supporting both top-down and bottom-up design processes. The
assembly model for embodiment product structures is at the top level of abstraction of product structures in a CAD/CAM system and can only support the bottom-up design process. GMM with its underlying concept is also promising in assembly automation for mechanisms. This is because the whole assembly process is actually divided into two parts: one responsible for the determination of kinematic positions (at the GMM level) and the other responsible for matching the volumetric object on the kinematic position. The work by Tilove (1983) actually provides a verification. He stated that the information of joint variables which could be calculated by general-purpose analysis systems can extend his work to the kinematic simulation of closed-loop mechanisms consisting of volumetric objects.

![Diagram of GMM and ISO-model's view](image)

*Figure 5.33. GMM vs ISO-Model in Describing Geometry of a Conceptual Body*
The Computer-Aided Mechanism Identification

6.1 Introduction

In the guideline to the general design approach by (NN, 1987), the recognition of a number of different solutions is considered as a very important product development activity. This activity is called mechanism identification in the context of the present study. Especially, the mechanism identification is based on the mechanism topology which deals with the structure regardless of magnitudes of any individual parameter of a machine body.

In the context of CIMOME the following scenario is possible. A tool called TADSOL generates a set of solutions, i.e. mechanisms and incidently none of them satisfies the requirement of a designer (an end-user). The designer tries to work out a new solution which is, however, just the same as those found by the computer tool. This may not be a problem to experienced users, however the system would be more intelligent if it can also help unexperienced users. A computer-aided identification of mechanisms is generally needed in this context. The identification of mechanisms based on topology is fundamental to the issue of structural analysis and synthesis of mechanisms (Yan & Hwang, 1991). The identification of mechanisms is also used as a tool to maintain a large design knowledge base such as a mechanism catalogue (Klein Breteler, 1983b) in such a way that any potential new mechanism must be checked whether it does not exist in the catalogue before it can become a member. The mechanism identification can also facilitate a team-based design scenario by maintaining the consistency of design information at two design sites where it is possible that the same mechanism is labelled differently—a potential factor of information inconsistency. The mechanism identification is also called unique representation of mechanisms and allows for mechanism topology comparison, regardless of labelling of bodies and joints.
In this chapter the mechanism identification is studied. Section 6.2 clarifies the concept of mechanism topology versus design stages. This founds the criteria for comparison of two mechanisms topologically. In Section 6.3 a new approach to the mechanism identification is described. Algorithms regarding this new approach have been developed and described in Section 6.4. Section 6.5 concludes this chapter.

6.2 Identification of Mechanism Structures on Different Abstraction Layers

6.2.1 Mechanism topology versus the design process
As implied in the earlier chapters the meaning of the mechanism structure evolves with the design process, and so does the mechanism topology. In other words, there should be different definitions of the mechanism topology at different abstraction levels where mechanism structures are viewed. The mechanism identification based on the mechanism topology is meaningful when a specific view of mechanism topology is determined. Here, four levels of views of mechanism topology are recognized.

**Mechanism Topology Level-1 (MTL-1 for short).**
At this level the type of the motion carrier (or body) is undefined, while joints are of basic types such as revolute joint, sliding joint, point-touch joint (*high pair* in planar mechanisms), etc. It is noted that gear joint and belt joint are not separately considered. Clearly, the viewpoint of this level of abstraction of mechanism topology is of motion-orientation.

**Mechanism Topology Level-2 (MTL-2 for short).**
At this level, with respect to the *MTL-1*, more specific joint types like gear joint and belt joint are separately considered. This viewpoint, compared with *MTL-1*, goes one step towards the object-orientation as *gear* and *belt* as well as *wheel* objects can be recognized implicitly.

**Mechanism Topology Level-3 (MTL-3 for short).**
At this level, with respect to the *MTL-2*, atomic-body types are distinguished and the fixed joint (or rigid joint) is introduced. In planar mechanisms, three categories of body types can be contemplated as shown in Fig.6.1. Clearly, the viewpoint of this level of abstraction of mechanism topology is of object-orientation except for the compound bar body in that its actual form depends on its connection with other bodies.

**Mechanism Topology Level-4 (MTL-4 for short).**
At this level, with respect to the *MTL-3*, the compound bar body is viewed as a set of bar bodies with rigid joints. The viewpoint of this level of abstraction of mechanism topology is of complete object-orientation.
Section 6.2 Identification of Mechanism Structures on Different Abstraction Layers

Machine-Atom-Body-Classification

gear, wheel, rack, etc. (well standard body)  cam (curved-body)  compound-bar (polygon body)

Figure 6.1. Classification of Machine Body Types at MTL-3

scheme (1)  scheme (2)

fixed on shaft

Figure 6.2. Two Schemes Differentiated at MTL-4

The information at MTL-1 and MTL-2 is generally operated within the functional design with the main concern being the motion transfer, e.g. the structural analysis and synthesis of mechanisms (kinematics). The information at MTL-3 and MTL-4 is generally operated within the context of the dynamic analysis and synthesis and embodiment design (shaping and layout of bodies). Especially the information at MTL-4 has also considered the identification of two schemes as shown in Fig.6.2 at the embodiment design stage. These two schemes should also be differentiated with respect to the dynamic behaviour when the offset distance along the axis perpendicular to the motion plane is taken into account. However, no topology information at these four levels can be developed, by adding geometry, to an embodiment (volumetric) mechanism, albeit the topology information at the level-4 is very close to a volumetric mechanism. From this viewpoint the configuration of these four information blocks in the framework of a product model, established earlier in Chapter 4, can be shown in Fig.6.3.

6.2.2 Identification for chain, mechanism, function generator and path generator

Most of the related work on the mechanism identification is meant for the kinematic chain identification. In the present study the kinematic chain identification has been extended to
Figure 6.3. Levels of Mechanism Topology in the Framework of Product Model

identify also the mechanism, e.g. function generator and path generator, etc. In particular, the function generator identification to CIMOME is important because a tool called TADSOL considered for integration is based on a mechanism catalogue. In the present study, the mechanism is the kinematic chain when a frame is defined. The function generator consists of two relative motions; their relationship is a kind of behaviour function. The path generator consists of a coupler; the motion of the coupler point produces a path.

6.3 An Approach to the Mechanism Identification

6.3.1 Basics of the mechanism identification

Definition 6.1. Line Graph. A mechanism is an assembly of bodies and joints. When a joint, say \( \alpha \) connects body \( i \) and body \( j \), two bodies are called adjacent and body \( i \) is called incident with joint \( \alpha \) and so does body \( j \). If a body is represented as a vertex (V for short) and an joint is represented a edge (E for short), a mechanism becomes a graph called line graph. Fig.6.4a shows an example.

Definition 6.2. Isomorphic graph. Two graphs say G1 and G2 are isomorphic when there is a one-to-one correspondence between the vertices of G1 and those of G2 with the property that the number of edges joining any two vertices of G1 is equal to the number of edges joining the corresponding vertices of G2, see Wilson (1972). So the identification of mechanisms is rendered to the identification of isomorphic graphs.
Figure 6.4. Definition of Line Graph and Adjacency Matrix for Kinematic Chains

**Definition 6.3. Adjacency Matrix (AM for short).** A line graph can be further represented in a matrix called adjacency matrix, AM = [a_lij], which is defined as:

\[
\begin{align*}
a_{ij} (i \neq j) & = 1, \text{ if } V_i \text{ and } V_j \text{ are adjacent} \\
a_{ij} (i \neq j) & = 0, \text{ if } V_i \text{ and } V_j \text{ are not adjacent} \\
a_{ij} (i = j) & = 0 \text{ or any other character (not significant).}
\end{align*}
\]

Fig. 6.4a also shows the AM representation of the graph.

**Definition 6.4. Weighted adjacency matrix.** If different joint types are distinguished then a weighted edge graph is obtained, see an example as shown in Fig. 6.4b. A weight number corresponds to a type of joint. Matrix AM can be extended to a weighted adjacency matrix (hereafter also AM for short), which is defined as:

\[
\begin{align*}
a_{ij} (i \neq j) & = \text{weight number, if } V_i \text{ and } V_j \text{ are adjacent.} \\
a_{ij} (i \neq j) & = 0, \text{ if } V_i \text{ and } V_j \text{ are not adjacent} \\
a_{ij} (i = j) & = 0 \text{ or any other character (not significant).}
\end{align*}
\]

Fig. 6.4b shows the AM which includes the information of joint types.

**Definition 6.5. Upper Triangular Adjacency Matrix (UTAM).** As noticed, AM is symmetrical with the diagonal filled with zeros. This means that all necessary information is in the triangle portion above the principal diagonal and is called Upper Triangular Adjacency Matrix (UTAM for short).

**Definition 6.6. Code of a line graph or an AM.** The code is a representation of the UTAM in a single line of digits in such a way that the rows of the UTAM are concatenated in a
sequence from top to bottom. For the two cases shown in Fig. 6.4, their codes are:

\[(101\ 10\ 1)_2\]
\[(21\ 1)_3\]

Fig. 6.4a
Fig. 6.4b

In the above the subscript number denotes the number system, e.g. '2' denotes the binary system. Such codes can further be converted to a decimal integer called code integer. The magnitude of this code integer depends on the labelling given to a graph. Since for a \(n\)-vertex graph there are \(n!\) different ways of labelling, the numbers of such code integers for one graph is \(n!\). In order to prove sufficiently that two graphs are non-isomorphic graphs, \(n!\) permutations of rows and columns are needed. (Hereafter this is called conventional way of permutation.) The objective to develop methods to identify isomorphic graphs is to avoid \(n!\) permutations.

**Definition 6.7. Degree of a vertex.** The degree of a vertex \(i\) (\(D_i\) for short) is defined as the number of edges incident to the vertex.

**Definition 6.8. Incident Degree of a vertex (ID for short).** The incident degree of a vertex was defined by Luo et al. (1991). Suppose vertex \(i\) in a graph has degree equal to \(t\) and is connected to its adjacent \(t\)-vertices with, respectively, weights \(W_{i1}, W_{i2}, \ldots, W_{it}\). Each adjacent vertex of vertex \(i\) has its own degree denoted by \(D_{i1}, D_{i2}, \ldots, D_{it}\). The incident degree of vertex \(i\) (ID, for short) combines these numbers in one code number:

\[ID_i = D_{i1}D_{i2}\ldots D_{it}W_{i1}W_{i2}\ldots W_{it}\]

by following the convention \(D_{i1} > D_{i2} > \ldots > D_{it}\). If some adjacent vertices have the same degree, then the weights of the corresponding edges should be in a descending order. Fig. 6.5 shows an example.

**Figure 6.5. Definition of Vertex Incident Degree**
Definition 6.9. Motion Property Graph (MPG for short). The information encoded in a weighted graph contains only the type of connections between two bodies. As an actual mechanism, such as the function generator and path generator, includes also the input/output information which is further represented as relative motion between two corresponding bodies, a motion property graph can be defined. Based on the line graph, motion properties (input, output or none) can be added as a kind of weight on the edges of the graph. The motion property graph can be represented by a Motion Property Adjacency Matrix (MPAM for short). A kind of code can be generated from the MPAM in the same way as code of the line graph (Definition 6.6). The motion property graph with its matrix representation was introduced first by Zhang (1991). Fig.6.6 shows an example.

\[
\begin{bmatrix}
0 & 1 & 1 & 0 \\
1 & 0 & 0 & 1 \\
1 & 0 & 0 & 1 \\
0 & 1 & 1 & 0 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
0 & 1 & 2 & 0 \\
1 & 0 & 0 & 0 \\
2 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
\end{bmatrix}
\]

*Note:* In MPAM, 1=input motion; 2=output motion; 0=none

**Figure 6.6. Example of Motion Property Adjacency Matrix**

6.3.2 A short review of the related work

A global overview of the methods described in the literature is presented in Fig.6.7, where corresponding references of each method are also shown. It is generally agreeable that for the kinematic chain (closed or open) identification, the characteristic polynomial method is only a necessary condition. The optimal code method is to find a canonical code number out of the n! permutations. There are two such codes possible: one having the smallest value of the code integer and the other having the biggest value. By making an analogy to the optimization technique, the unconstrained optimal code method corresponds to a scheme to find the minimum (or maximum) code without imposing any restriction on the vertices of a concerned graph. The constrained optimal code method imposes some restrictions on the vertices of a graph in terms of their features, such as the degree of the vertices, and performs conventional permutations, e.g. the method by Tang & Liu (1988). (The terms constrained code method and unconstrained code method are introduced here with the objective to explore the nature of various code methods in the literature.) The unconstrained optimal code method needs complicated algorithms and extensive computations and still leaves some problems open, see the comments by Ambekar & Agrawal (1987a) and Tang & Liu (1988). The constrained optimal code methods so far reported in the literature can usually reduce the n! permutations to a(\!b!...\!h!\), where a,b,...,h are the numbers of the vertices with the same feature and a+b+...+h=n. However, a problem occurs when all the vertices have the same
feature, e.g. a pivoted four bar chain or one big group with the same feature. Ambekar & Agrawal (1987b) also extended their unconstrained code method to the identification of mechanisms, function generators and path generators (some remarks are given in Section 6.5).

\[
\begin{align*}
\text{the kinematic chain identification methods} & \quad \text{based on line graph} \quad \text{based on dual graph} \\
\text{characteristic polynomial} & \quad \text{optimal code} \quad \text{Sohn & Freudenstein (1986)} \\
(Uicker & Raicu, 1975; Yan & Hall, 1982) & \quad \text{Al-Hakim & Shrivastava (1991)} \\
\text{unconstrained optimal code} & \quad \text{constrained optimal code} \\
(Randic, 1974; Ambeka & Agrawal, 1987) & \quad \text{(Tang & Liu, 1988) } \quad \text{incident code (Luo, et al, 1991)}
\end{align*}
\]

*Figure 6.7. An Overview of the Related Work on Kinematic Chain Identification*

**6.3.3 The proposed approach**

The proposed approach to the mechanism identification in the present study is based on the constrained optimal code method. In particular the approach contains two considerations. The first consideration is to extend the Incident Degree to the Vertex Feature Degree (VFD for short) which is a new idea and will be illustrated in more detail later in this section. The second consideration is to develop an algorithm which performs permutations of rows and columns within a group of vertices with the same VFD, see the next section. This second consideration differs from the unconstrained code methods in that they work on the whole region of the vertices. The second consideration will evidently overcome the drawback of the constrained code method in that the number of permutations in a group of vertices with the same VFD won't be a! (a = the number of vertices with the same VFD).

It is noted that the underlying philosophy of both the degree code method and the incident degree code method is actually to extract features of an individual vertex, say i. The degree code method extracts the degree of a vertex as a kind of feature. In a weighted graph, the weight of the incident edges is of course another kind of feature. This implies that there should have been another kind of degree that contains both the information of the vertex degree and the weight of the incident edges, albeit so far this kind of degree has not been discussed yet. The incident degree takes more features: besides the two features mentioned earlier it includes degrees of its adjacent vertices as a kind of feature of the vertex currently being concerned, see *definition 6.7* and *definition 6.8* respectively. Along this route of thinking it can be found that the source of extracting features of an individual vertex, say i,
Section 6.3  An Approach to the Mechanism Identification

can generally be both vertex i itself and its environment (i.e. incident edges and adjacent vertices), since a vertex is not isolated. From this viewpoint, the degree code method only extracts the feature of vertex i itself while the incident degree code method also extracts the features of the environment of vertex i. The VFD proposed in the present study extends the incident degree and is defined below.

Assume that vertex i has t adjacent vertices. We define for vertex i:

K-VFDi := \text{BT}_i \downarrow \text{BT}_i \text{BT}_2 \ldots \text{BT}_t | \text{D}_i \text{D}_2 \ldots \text{D}_t | \text{W}_i \text{W}_2 \ldots \text{W}_t \quad \text{(for the kinematic chain)}

M-VFDi := \text{Frame} | \text{BT}_i | \text{BT}_i \text{BT}_2 \ldots \text{BT}_t | \text{D}_i \text{D}_2 \ldots \text{D}_t | \text{W}_i \text{W}_2 \ldots \text{W}_t \quad \text{(for the mechanism)}

F-VFDi := \text{Frame} \downarrow \text{BT}_i | \text{BT}_i \text{BT}_2 \ldots \text{BT}_t | \text{D}_i \text{D}_2 \ldots \text{D}_t | \text{W}_i \text{W}_2 \ldots \text{W}_t | \text{MP}_i \text{MP}_2 \ldots \text{MP}_t \quad \text{(for the function generator)}

P-VFDi := \text{Coupler} | \text{Frame} | \text{BT}_i | \text{BT}_i \text{BT}_2 \ldots \text{BT}_t | \text{D}_i \text{D}_2 \ldots \text{D}_t | \text{W}_i \text{W}_2 \ldots \text{W}_t | \text{MP}_i \text{MP}_2 \ldots \text{MP}_t \quad \text{(for the path generator)}

where

- \text{BT}_i \quad \text{denotes the body type of vertex i.}
- \text{BT}_j \quad \text{denotes the body type of vertex j which is adjacent to vertex i; for the implementation consideration a number is assigned to each type of body.}
- \text{MP}_j \quad \text{denotes the relative motion properties between vertex i and its adjacent vertex j, see definition 6.9; for the implementation consideration a number is related to each type of motion property such as \(1=\text{input, 2=output and } 0=\text{none.}\)}
- \text{Frame} \quad \text{denotes whether or not vertex i corresponds to the frame body; for the implementation consideration, symbol '1' denotes that the vertex is a frame body otherwise symbol '0' is assigned.}
- \text{Coupler} \quad \text{denotes whether or not vertex i corresponds to a coupler, a point of which produces a path; for the implementation consideration, symbol '1' denotes that the vertex concerns a coupler body otherwise symbol '0' is assigned.}
- \text{Dij} \quad \text{denotes the degree of the adjacent vertices of vertex i; for the implementation consideration, a number is related to this feature.}
- \text{Wij} \quad \text{denotes the weights of the incident joints of vertex i; for the implementation consideration, a number is related to this feature, e.g. 1=revolute joint, 2=sliding joint, 3=gearing joint, etc.}

Furthermore in the definition of the VFD, the parts with an underlined reflect the feature of vertex i itself while the other parts reflect the features of the environment of vertex i. In the following, several examples are presented. When the mechanism identification is performed at different levels of mechanism topology as earlier defined, non-relevant parts may be omitted. For example, for the identification at MTL-1 or 2, part (\text{BT}_i | \text{BT}_i \text{BT}_2 \ldots \text{BT}_t | ) is omitted.

\text{An optimal code based on VFD (called VFD-code for short) is a code based on a condition}
on vertices such as: all vertices are arranged in a descending order of their VFDs. (In the implementation, a vertex which represents the frame is always labelled as nr.1.) In terms of the levels of topology for identification and the sorts of identified entities, i.e. the kinematic chain, the mechanism, the function generator or the path generator, a VFD-code has the following forms:

VFD-code-1 (for the kinematic chain at MTL-1 or 2) ::=< maximal code generated from AM >.
VFD-code-2 (for the kinematic chain at MTL-3 or 4) ::=< maximal code generated from AM, code generated from a body list >
(A body list would be, e.g. (1,2,1,3,1) (a graph with five vertices), where the number corresponds to the type of an individual body. A code (a decimal integer) can be generated from (12131), for example).
VFD-code-3 (for the mechanism at MTL-1 or MTL-2) ::=< maximal code generated from AM >.
VFD-code-4 (for the mechanism at MTL-3 or MTL-4) ::=< maximal code generated from AM, code generated from a body list >.
VFD-code-5 (for the function generator at MTL-1 or MTL-2) ::=< maximal code generated from AM, code generated from MPAM >
VFD-code-6 (for the function generator at MTL-3 or MTL-4) ::=< maximal code generated from AM, code generated from a body list, code generated from MPAM >
VFD-code-7 (for the path generator at MTL-1 or MTL-2) ::=< maximal code generated from AM, Ci, code generated from MPAM >
(Ci=the vertex number corresponding to the coupler after the maximal code from AM is generated.)
VFD-code-8 (for the path generator at MTL-3 or MTL-4) ::=< maximal code generated from AM, Ci, code generated from body list, code generated from MPAM >

Example-1 (Fig.6.8). Fig.6.8a,b are kinematic mechanism diagrams of kinematic chains and Fig.6.8c shows both the graph and the adjacency matrix representations. The question is to generate VFD-codes.

(i) Identification at MTL-2. Fig.6.8c shows the line graph and the AM at the initial labelling. All K-VFDi (i=1,2,...,5) are as follows:

<table>
<thead>
<tr>
<th>vertex nr</th>
<th>degree of vertex</th>
<th>K-VFD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>2211</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2231</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2221</td>
</tr>
</tbody>
</table>

132
Using the degree code method, 5! permutations are required because the five vertices have all the same degree. Using the VFD-code method, 2!2! permutations are required (notice that here an optimal algorithm is not yet applied, it will be introduced in the next section). Arrangement of vertices in a descending order of VFDs (i.e. permutations: 1↔4 and 4↔5, where ‘↔’ denotes interchange of two vertices) produces the maximal AM:

\[
\begin{bmatrix}
0 & 3 & 1 & 0 & 0 \\
3 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 2 & 0 \\
0 & 0 & 2 & 0 & 1 \\
0 & 1 & 0 & 1 & 0
\end{bmatrix}
\]

Further permutations within groups of vertices with the same VFD may be needed. However, for this example no permutation can further yield a bigger AM.

The VFD-code = \( <(3100001201)_4> \) in terms of the VFD-code-1 defined above.
(ii) Identification at MTL-3 or MTL-4. All K-VFDi (i=1,2,…,5) are as follows:

<table>
<thead>
<tr>
<th>vertex nr</th>
<th>degree of vertex</th>
<th>body-type</th>
<th>body-type-character</th>
<th>K-VFD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>bar</td>
<td>1</td>
<td>1212211</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>gear</td>
<td>2</td>
<td>2312231</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>bar</td>
<td>1</td>
<td>1312212</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>rack</td>
<td>3</td>
<td>3212231</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>bar</td>
<td>1</td>
<td>1112221</td>
</tr>
</tbody>
</table>

None of the vertices have the same VFD. Arrangement of vertices in a descending order needs one permutation, i.e. 1↔4 and yields:

\[
\begin{bmatrix}
0 & 3 & 1 & 0 & 0 \\
3 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 2 \\
0 & 1 & 0 & 0 & 1 \\
0 & 0 & 2 & 1 & 0
\end{bmatrix}
\]

\[
\text{AM} = \begin{bmatrix}
0 & 3 & 1 & 0 & 0 \\
3 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 2 \\
0 & 1 & 0 & 0 & 1 \\
0 & 0 & 2 & 1 & 0
\end{bmatrix}
\]

The VFD-code = \(<(3100010021)_4, (32111)_4>\) in terms of the VFD-code-2. Suppose that body nr.4 (initial label), the rack, is replaced by a gear (Fig.6.8b). All K-VFDi (i=1,2,…,5) are then as follows:

<table>
<thead>
<tr>
<th>vertex nr</th>
<th>degree of vertex</th>
<th>body-type</th>
<th>body-type-character</th>
<th>K-VFD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>bar</td>
<td>1</td>
<td>1212211</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>gear</td>
<td>2</td>
<td>2212231</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>bar</td>
<td>1</td>
<td>1212212</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>gear</td>
<td>2</td>
<td>2212231</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>bar</td>
<td>1</td>
<td>1112221</td>
</tr>
</tbody>
</table>

The VFDs of vertices nr.2 and nr.4 are now the same. Arrangement of vertices in a descending order needs one permutation, i.e. 1↔4 and leads to the same AM as the one before. Although vertices 1 (originally 4) and 2 have the same VFD, further permutation won’t make a bigger AM. The VFD-code = \(<(3100010021)_4, (221111)_4>\) in terms of the VFD-code-2. It has been shown that without consideration of the body type, the two cases (Fig.6.8a and Fig.6.8b) are of the same topology. Therefore it is necessary to have the definition of VFD-code-2, where the information about the body type of each vertex in a graph is also included. It is very important to notice that the code generated from only the AM does not include the body type information, albeit it is affected by the body type information. Inclusion of the body type information in the code is necessary to generate a
Section 6.3  
An Approach to the Mechanism Identification

complete kinematic mechanism diagram from the code, which was considered to be an important advantage of the code methods, see Tang & Liu (1988).

![Path Generator, AM and MPAM for Example-2](image)

**Figure 6.9. Path Generator, AM and MPAM for Example-2**

**Example-2** (Fig. 6.9a). Fig. 6.9a shows the kinematic mechanism diagram of a path generator with two inputs and Fig. 6.9b shows both its line graph with its adjacency matrix and the motion property graph with its motion property matrix. (In the motion property matrix the feature of a coupler is placed on a corresponding diagonal element for simplification of the identification procedure.) The question is to generate its VFD-code in terms of MTL-1 or 2.

In terms of the definition of the P-VFD, the P-VFDi (i=1,2,...,5) are as follows:

<table>
<thead>
<tr>
<th>vertex nr</th>
<th>degree of vertex</th>
<th>P-VFD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>0 1 22 11 10</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0 0 22 11 10</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>0 0 22 11 10</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1 0 22 11 00</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>0 0 22 11 10</td>
</tr>
</tbody>
</table>

where, as mentioned earlier, vertex 1 is related to the frame body and vertex 4 is related to a coupler. Notice that the first index (nr.1) is always assigned to the frame body, only one permutation (2e→4) is needed to arrange the VFDs in a descending order. This will yield AM1 and MPAM1 as shown below:

135
\[
\begin{bmatrix}
0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 \\
1 & 0 & 0 & 0 & 1 \\
1 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0
\end{bmatrix}
\]
\[
\begin{bmatrix}
0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 0
\end{bmatrix}
\]

AM1 = MPAM1 =

Since the last three vertices have the same VFD, further permutations may be needed to obtain a maximal AM. This is done by interchanging vertex 3 and vertex 4 and leads to the AM2 and MPAM2 below.

\[
\begin{bmatrix}
0 & 0 & 1 & 1 & 0 \\
0 & 0 & 1 & 0 & 1 \\
1 & 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 1 & 0
\end{bmatrix}
\]
\[
\begin{bmatrix}
0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 0
\end{bmatrix}
\]

AM2 = MPAM2 =

The VFD-code = \<(0110011010)_2, 2, (0100000001)_2, > in terms of the VFD-code-7.

6.4 Algorithms

As mentioned in the previous section, the proposed approach needs an algorithm to perform permutations within a group of vertices with the same VFD when such groups exist. This algorithm has been developed and is briefly explained in the following. The general philosophy of the algorithm is to settle down labels for those vertices with the same VFD. The goal is to make the AM bigger. The general method to make AM bigger includes two considerations: (1) to arrange a bigger weight number to the left as far as possible on a specific row and (2) to arrange a bigger weight number to the top as far as possible on a specific column. (Notice the definition of VFD-code in Section 6.3.3 and definition 6.6.) When VFDs of a graph are arranged in a descending order, a corresponding AM is divided into different areas which reflect the relationship among vertices. Therefore it will be helpful to study influences on these areas with respect to interchanging two vertices in one group with the same VFD.
6.4.1 The map of the AM
After VFDs of a graph are arranged in a descending order, different areas of an AM are formulated like a 'map'. Fig.6.10 shows a general map with two groups of vertices with the same VFDs. The different groups are denoted by (i1,k1) and (i2,k2), respectively, where i1 and i2 denote the labelling of the first vertex of group-1 and group-2, while k1 and k2 denote the number of vertices within group-1 and group-2. This figure shows the five feature areas denoted, respectively, by FA-1,FA-2,...,FA-5. FA-1 is the area about the connection information of the vertices of group-1 with the vertices which have bigger VFDs than those in group-1. FA-2 is the area about the connection information of the vertices of group-1. FA-3 is the area about the connection information of vertices of group-1 with the vertices which have smaller VFDs than those in group-1. FA-4 is the area about the connection information of two such groups. FA-5 is the area similar to FA-3. Several propositions can be concluded from this figure.

![Map of AM with VFDs in Descending Order](image)

**Legend:**

- **FA-i:** Feature Area Nr:i,
- - denotes the areas that will not be disturbed when switching among vertices from i1 to i1+k1-1

**Figure 6.10 Map of AM with VFDs in Descending Order**

**P-1.** The labelling of the vertices (i1, i1-1), (i1+k1, i2-1) and (i2+k2, n) has been settled down, where n = the number of vertices in a graph. This is so because of their unique VFD. No permutations should affect their positions among vertices.

**P-2.** Any permutation on the vertices before i1 makes no sense with respect to our philosophy and goal as well. This is obvious because of P-1. When all the vertices in group-1 are settled down this will further be applied to group-2, i.e. to vertex i2.

**P-3.** Any permutation within FA-3 should be done only by switching columns from i1 to i1+k1-1. This is because labels of vertices within (i1+k1, i2-1) have been already settled down (see P-1 above). Permutations in this area change the connections of vertices of group-1 and those vertices with smaller VFDs.

**P-4.** Any permutation within FA-5 is similar to that in FA-3 with only different borders.

137
6.4.2 Algorithm
The algorithm will be used both for creating a VFD-code and comparing two mechanisms topologically. For the latter's purpose, preliminary comparisons are first done, like checking some necessary conditions (for two mechanisms to be the same) e.g. the number of bodies, the number of joints, etc. After the necessary conditions are checked and passed, the following algorithm will be initiated. The algorithm can be outlined with the following main steps. A PASCAL-like pseudo code is used for the description of this algorithm. (Throughout the dissertation this pseudo code will be used whenever descriptions of algorithms are needed.) The conventions of this pseudo code formalism are as follows

- Uppercase words are key words of the pseudo code, e.g., WHILE...DO, etc.
- Logical symbols are used in the description, such as '⇒' denotes 'assign to', etc.
- Italic words are verbs dependent of applications
- Texts within the bracket symbol '{...}' are comments.
- Strings within '...' are variables dependent of applications

Assume:

1. 'cr' denotes the current row number; 'sr' denotes the starting row number.
2. 'cc' denotes the current column number; 'sc' denotes the starting column number.
3. 's(cr)' denotes the number of highest weight numbers on the far left on the current row.
4. The notations used in Fig.6.10 are further applicable to the following description.

Algorithm outline:

Step-1 (permutations in FA-1).

\[ i1 \Rightarrow 'sc'; \quad I \Rightarrow 'cr'; \quad k1 \Rightarrow s(cr); \]
\[ \text{WHILE} \left( 'sc' \neq i1 + k1 - 1 \right) \quad \text{DO} \{ \text{post-condition: not all vertices in a group are labelled} \} \]
\[ \text{WHILE} \left( 'cr' \neq i1 \right) \quad \land \quad \left( 's(cr)' > 1 \right) \quad \text{DO} \]
\[ \quad \text{switch the highest weight number on 'cr' between the columns from} \]
\[ \quad 'sc' \text{ to 'sc' + 's(cr)' - 1 from the right to the left;} \]
\[ \quad \text{count the number and record it into 's(cr+1)'}; \]
\[ \quad 'cr' + 1 \Rightarrow 'cr' \]
\[ \text{END WHILE} \]
\[ \text{IF} \left( 's(cr)' > 1 \right) \quad \land \quad \left( 'cr' = i1 \right) \text{ THEN} \]
\[ \quad \text{do permutations in FA-2} \{ \text{see Step-2 of the algorithm} \} \]
\[ \text{END IF} \]
\{Post-condition: it has been decided which vertex among group-1 should be labelled as number 11\}
\[ 'sc' + 1 \Rightarrow 'sc' \quad \{ \text{A new column is decided for possible further permutations in group-1.} \} \]
1 \Rightarrow \text{'cr'}; \quad \{A \text{ new round of searching is also started from the first row.}\}

k1+i1-sc \Rightarrow \text{'s(cr)'} \quad \{\text{The size of group-1 is reduced now because one of vertices has obtained the label number i1.}\}

END\_WHILE

\[ s=3 \]

\begin{itemize}
  \item h1 corresponds to the column encoded in 'sc' after the permutation in FA-1 has failed.
\end{itemize}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fa-2-permutations}
\caption{Figure 6.11. Permutations within FA-2}
\end{figure}

\textbf{Step-2 (permutations within FA-2)}

In this area, a switch between two columns (on the current row) is not only affected by the weight numbers on the columns but also by the weight numbers on the corresponding rows (notice the symmetrical property in this area), see Fig.6.11 (assume the amount of vertices with arbitrary numbers is '3'). So a different approach must be taken. In the first place the amount of the vertices concerned in this area must be more than 3; otherwise a switch does not make any sense (the adjacency matrix is symmetrical). In the second place the vertex which has the highest weight number must be arranged in the place of the row corresponding to the starting column encoded in the variable 'sc' (see \textit{Step-1} above) and the column just one on the right of the diagonal element. In the case of Fig.6.11, the weight number '3' can be arranged in that place by switching the column ('sc') (i.e. number h1 in Fig.6.11) and the column ('sc'+3-1) (i.e. number h3 in Fig.6.11). Permutations in FA-2 may still not resolve the issue which vertex should be placed on the column encoded in the variable 'sc' (see \textit{Step-1} above). This implies that there are still \(s(>1)\) vertices which can be placed on the column ('sc') while the magnitude of AM is kept the same. If so, the algorithm goes to FA-3 to test whether the issue can be resolved, see \textit{Step-3}.

\textbf{Step-3 (permutations within FA-3)}

It is noted that when column ('sc'+s'-1) < column (i1+k1-1), the searching actually does not reach to FA-3, see Fig.6.12a. Since the vertices between i1+s and i1+k1-1(i1 < i1' < i1 + k1-
1, see Fig.6.12a) have not been settled down yet, the permutation in this area should be postponed till any vertex of the vertices between \( i1' + 's' \) and \( i1+k1-1 \) is settled down. This means that the unresolved vertices between \( i1' \) and \( i1' + 's'-1 \) must be recorded and be further considered whenever an opportunity arises, e.g. any vertex between \( i1' + 's' \) and \( i1+k1-1 \) is settled down. So the algorithm will skip these postponed vertices. A complex situation may happen. When determining the labelling for the vertices between \( i1' + 's' \) and \( i1+k1-1 \), the issue cannot be resolved till the searching reaches the row \( i1' \) and \( i1' + 's'-1 \) (postponed vertices). This is the time that two unresolved groups meet and simultaneous consideration is needed; the algorithm goes to Step-4. When \( i1' + s-1 = i1 + k1-1 \), i.e. the permutation does reach \( FA-3 \), in terms of \( P-3 \) (see Section 6.4.1), the algorithm in Step-3 is outlined below.

![Diagram](image)

**Figure 6.12. Simultaneous Consideration of Unresolved Labelling Areas in Step-3**

\[ \text{il} + k1 \Rightarrow 'cc'; \]
\[ 's' \Rightarrow 'cs' \]
\[ 'cs' \text{ encodes the current amount of unresolved vertices in the current step.} \]
\[ 's' \text{ encodes the amount of unresolved vertices after permutations of Step-2.} \]

\[ \text{WHILE ('cc' < i2) \&\& ('cs' > 1) DO} \{ \text{The permutation is still in } \text{FA-3.} \} \]

\[ \text{\textit{find} the highest weight number on column 'cc' from row 'sc' to row 'sc' + 's'-1} \]

140
record the amount of vertices with the highest number into 'cs'

arrange the vertices with the highest weight number to the far top

'cc'+1 ⇒ 'cc' {move to the next column.}

END WHILE

IF ('cs' > 1) THEN {post-condition: the unresolved issue cannot be solved in FA-3.}

skip over FA-4 {This is because the vertices between i2 and i2+k2-1 are themselves to
be settled down, the same reason as that mentioned earlier in Step-3.
When the labelling for group-2 goes to FA-4 and there exists an unresolved
issue in a previous group, say group-1, the simultaneous
consideration of two groups are needed, see Step-4. }

go to FA-5 { Step-5}

END_IF

Step-4 (the simultaneous consideration of two unresolved groups)

Taking the situation shown in Fig.6.12a for the illustration purpose, the algorithm tries to
produce a situation that will make AM bigger by arranging these weight numbers. In the case
of Fig.6.12b the algorithm produces that situation as shown. It is clear that this situation has
resolved the postponed unresolved issue in group-1 and at the same time the current
unresolved issue as well, see the discussion in Step-3.

Step-5 (permutation in FA-5)

In terms of P-4 (see Section 6.4.1), this is the same as Step-3 by setting the starting column
as i2+k2.

6.5 Discussion and Conclusions

The main observation arising from the present study with respect to the mechanism topology
identification is the necessity to study the issue of the mechanism topology identification in
the context of a design process. This leads to the definition of several levels of mechanism
topology. For example it would be necessary to compare two mechanisms with consideration
of body types but still dimensions of a body being neglected. In general a computer support
for identification is needed in the context of CIMOME.

Based on two surveys provided by Zhang (1991) and Jongsm (1991), it has been found that
the methods in the current literature are not sufficient to our requirements. For example the
identification of function generators and path generators was not well discussed and there has
been no method available to identify a mechanism in terms of different design levels.
Furthermore, those algorithms have some limitations (see also Section 6.3.2).
A new approach is proposed, which can be applied to the identification of kinematic chains, mechanisms, function generators and path generators with multiple degrees of freedom and with different levels of mechanism topology in terms of a design process. Compared with the approach proposed by Ambekar & Agrawal (1987b) for the identification of mechanisms, function generators and path generators, additional codes for the extra information about mechanisms, function generators and path generators were established in their method. Extra permutations are needed to produce an optimal code for those extra codes. In our proposed approach the extra information is represented as a kind of features on vertices, for which no any extra permutation is necessary.

An experimental computer program based on this new approach with the outlined algorithm has been developed by Jongsm (1992a) and Jongsm & Zhang (1992b). The program was tested to detect isomorphism for two special kinematic chains with 28 bodies and all revolute joints, as shown in Fig.6.13b,c by their graphs. These two graphs are originated from the graph of Fig.6.13a by removing any pair of vertices connected, as indicated. For giving an impression of the complexity and the size of the problem, Fig.6.14a,b shows their adjacency matrices with an arbitrary initial labelling. Both of two graphs (thus kinematic chains) have three groups of the same VFDs: (1) group-1: 16 vertices with the same VFD as 333111, (2) group-2: 8 vertices with the same VFD as 332111 and (3) group-3: 4 vertices with the same VFD as 3311. The computer program (running on a 486 personal computer) takes a few seconds to get the report: the two graphs are isomorphic. The result was in accordance with Dijksman & Timmermans (1992). Using the incident code method by Luo et al. (1991), it has to perform \((16!8!4!)\) permutations which might take a whole weekend for this computer.
Figure 6.13. A Test Case for the Algorithm
Editors in CIMOME

7.1 Introduction

An editor is used in the computer program development. It is a tool to write program source codes for further processing, e.g. compiling, debugging, etc. By making an analogy of the machine structure and the program source code, the term Editor is also used for the computer-aided machine design. It is then clear that the editor is in the first place related to the mechanism structure. The editor in the modern CASE environment is used not only to write the source code but also to specify the requirement which is subsequently used as inputs of certain tools to generate source codes. Similarly, in CIMOME editors are classified as:

- **class (i)**: the editor for the machine artifact specification.
- **class (ii)**: the editor for the requirement specification.

As mentioned earlier in Chapter 4 and Chapter 5 both artifact structures and requirements are divided into various levels corresponding to design stages. Furthermore structures and requirements might be viewed from various aspects that individual design tools require and operate on, see Section 4.3 of Chapter 4. This leads to an organizational structure of editors and design tools, see Fig.7.1. This figure implies a concept of generic editors and specialized editors. The generic editor is served for more than one tool while the specialized editor is a need for certain individual tools. A typical example of application of this concept is in integrating a tool called OPLAM into CIMOME (see also Chapter 9). OPLAM is a tool to use the optimization technique for the dimension synthesis of mechanisms (see also Chapter 2). For this tool to work, first a type of mechanism must be specified and then certain parameters of the mechanism must be specified as design variables of an optimization problem. The task of the former belongs to the category of the generic editors while the task
of the latter belongs to the category of the specialized editors. In general, editors are parts of the user-interface for various tools in the context of the CIMOME architecture.

![Diagram of Editors in CIMOME]

*Figure 7.1. Classification of Editors in CIMOME*

In this chapter a generic editor for conceptual mechanism structures (the system is called EDMECH for short) is described. The system is actually a user-interface for GMM described earlier in Chapter 5. The development goal of this editor is to achieve as much as possible user-friendliness and thus to raise the intelligence of CIMOME. The main approach is to follow the modern user-interface system developing methodology (Foley *et al.*, 1990). In particular a requirement analysis of EDMECH is presented in Section 7.2. The architecture of EDMECH is discussed in Section 7.3. The implementation of a prototype system is presented in Section 7.4, in which a general purpose UIMS developed with CIMOME is introduced as well. Section 7.5 concludes this chapter.

7.2 A Requirement Analysis of the EDMECH System

An editor system is an interactive system. An editor system is intelligent when its external behaviour is user-friendly. To achieve as much as user-friendliness an analysis of the mental model of potential end-users is necessary. It is proposed in the present study that the mental model can be structured into two parts: generic part and specialized part. The generic part is common to all user interface systems and it includes

- an end-user does as few operations as possible while a system does more,
- an end-user likes to immediately see any consequence on the screen just after his operation (*what you see is what you get* philosophy),
- help messages must appear just when an end-user needs it (*just-in-time* philosophy in the man-machine interaction), and

146
Section 7.2  

Requirement analysis of the EDMECH system

- the number of operations to construct an application object is minimal (short-cut way capability).

The last aspect is also related to the issue how a broad spectrum of potential end-users (from those with a little knowledge of an application to those with ample knowledge) are accommodated. Experienced end-users may use some short-cut ways provided by the system to construct application objects more fast.

The specialized part depends on a particular application in the sense that a set of assumptions of the knowledge of potential end-users is required. In the mechanism application it includes the knowledge of mechanism objects, the knowledge of the graphic representation of mechanism objects and the knowledge of procedures to construct mechanism objects. EDMECH takes the following assumptions:

- The end-user knows that a mechanism is composed of a set of machine parts, e.g. gear, wheel, rack, bar, etc. and a set of joints, e.g. revolute, sliding, rigid, etc. The knowledge described in Section 5.4 of Chapter 5 about the conceptual mechanism structure is a priori known to the end-user. (This implies that GMM described in Chapter 5 will be an underlying object model for EDMECH.)
- The procedure of graphically building a mechanism follows the way of sketching a mechanism on a sheet by human designers. Fig.7.2a shows a mechanism to be constructed in a designer's mind and Fig.7.2b shows two possible procedures that the end-user may prefer.

![Diagram of a mechanism sketch and procedures](image)

*Figure 7.2. Two Procedures of Drawing a Mechanism Sketch*

The discussion above sets up the requirement on developing EDMECH at the aspect of user friendliness. Regarding the aspect of the functionality of EDMECH the following requirements are envisaged:

- The EDMECH should allow for graphically constructing the usual types of machine parts
(bar, gear, cam, wheel, etc.) and types of joints (revolute, slide, etc.).

- The EDMECH should allow for possible different graphic representations for the same semantics to achieve a comprehensive and straightforward representation. For example it should allow for the representation of a sliding joint between two bodies in various ways as shown in Fig. 7.3. Fig. 7.3 also shows the different graphic representations for a bar body, i.e. a simple straight line segment, a long box and a short box.

![Various graphical representations of a bar body](image)

*Figure 7.3. Various Possible Graphic Representations of Sliding Joint and Bar Body*

7.3 The Architecture of EDMECH

First, several definitions regarding a Graphic User Interface (GUI for short) system are given. The architectures of EDMECH are then viewed in two aspects.

7.3.1 Basic concepts of a user-interface system

**Definition 7.1. Gadget.** A gadget is a means for user-computer interaction (Hardwick, 1990). Examples of gadgets are: menus, text entry boxes, dialogue boxes and display boxes for application-generated images (in the present study, a mechanism). A gadget can be further divided into a set of sub-gadgets, just as a task can be decomposed into sub-tasks. Non-decomposable gadgets are called widgets.

**Definition 7.2. Window.** A window is a rectangle box through which the functionality of a gadget is made explicit. It is clear that windows are a kind of vehicles to carry the semantics of gadgets. A window has various forms and is placed on a certain position of the screen.

**Definition 7.3. Windowing system.** It provides the basic tools needed to create windows, display graphics and respond to the mouse as it moves about the display screen (Scheifler & Gettys, 1986). Windowing systems contain many of the features found in graphics packages, for details see Hardwick et al. (1990). In the present study, graphics packages are assumed
to be built in a windowing system.

**Definition 7.4. User-Interface.** A *user-interface contains a set of gadgets*. Sometimes gadgets are also called *user-interface elements* in a more broad sense. A user-interface consists of a set of user-interface elements.

**Definition 7.5. State-net.** Obviously user-interface elements are related to each other. A state of the user-computer interaction is reflected in a current active user-interface element. The state transition history forms a *state-net* or *state transition diagram* (Foley et al., 1990) which specifies the relationship of the user-interface elements.

**Definition 7.6. User Interface Management System (UIMS).** A *UIMS is a tool for the programmer or casual user to create and manage user interfaces for application programs* (Hardwick et al., 1990). The components of a UIMS are:

1. a windowing system, see Definition 7.3,
2. a toolkit for creating gadgets, and
3. a dialogue control system.

The dialogue control system is used to describe the flow and the content of an interface. For example, if a pop-up menu is considered, the toolkit allows us to create this menu, but the dialogue control system describes when the menu is displayed to the end-user, what the contents of that menu should be and what should happen when each menu items is selected. A control system usually contains three components:

1. a *gadget manager*
2. an *application manager*, and
3. an *executive manager*.

![Figure 7.4. Relationship of Components in a UIMS Control System](image)

The *gadget manager* is responsible for selecting the right gadget when a user-interaction event happens. The *application manager* is responsible for communication with an application, e.g. sending a message to a certain application routine for some calculations. The *executive manager* takes care of the state-net and the communication with other two managers. Fig. 7.4
shows the relationship among these three components.

![Diagram](image)

**Figure 7.5. Architecture of EDMECH in Viewpoint of Functionality**

### 7.3.2 The Architecture of EDMECH in the function's view

From the function's point of view three system components are identified: a User-InterFace Manager (UIM), a Mechanism Presentation Manager (MPM) and a Mechanism Object Manager (MOM), see Fig.7.5. The UIFM is responsible for the tasks:

1. displaying a mechanism presentation (or graphic representation) and
2. executing the user-computer dialogue in terms of a predefined specification via such as *transition network diagram* (Foley et al., 1990).

![Diagram](image)

**Figure 7.6. The Mechanism Presentation Model: a Conceptual View**
The MPM manages the mechanism presentation model and the MOM manages the database model of conceptual mechanism structures, i.e. GMM (Chapter 5). The mechanism presentation model is shown in Fig.7.6 and it represents the information:

(1) pictures of each mechanism object, see Fig.7.7, and
(2) the connection between mechanism objects stored in GMM and corresponding graphic metaphors.

![Diagram showing mechanism objects and connections](image)

*Figure 7.7. Examples of Pictures of Bodies in EDMECH*

The procedures of two generic operations in the system are described in the following to show how the three managers collaborate.

*Creation* of (a machine part *bar*).

Step-1: An end-user presses the *create* button (see Fig.7.7), which causes UIFM to trigger
a user-computer dialogue session to get the end-user's intents for the graphic form *a long box* (Fig.7.3c).

**Step-2:** UIFM displays a long box with the help of a display toolkit in a windowing system (see in Fig.7.7 the menu on the right).

**Step-3:** The graphic information of this bar is written into corresponding classes of the mechanism presentation data model via MPM.

**Step-4:** The geometry of the bar, such as the length of the bar is written into corresponding classes of GMM via MOM.

*Deletion of* (a machine part *bar*).

**Step-1:** An end-user presses the *delete* button (see Fig.7.7), which causes UIFM to trigger a user-computer dialogue session to guide the user to select a bar via *e.g.* the mouse.

**Step-2:** UIFM determines which machine parts are pointed by the mouse. At this time searching is only performed on the mechanism presentation model via MPM.

**Step-3:** After a machine part is selected and removed out of the screen, instances in corresponding classes of GMM are deleted via MOM.

There are two advantages to set a mechanism presentation layer between the end-user and the GMM. *Firstly*, forms of a specific presentation won't influence underlying mechanism objects. The presentation is essentially a kind of *window* through which end-users can look at a mechanism object. The different presentations on the same mechanism object are quite likely as a unified standard is still under investigation. *Secondly*, this layer of information makes possible an explicit communication between the end-user and the computer on the pictures of machine objects, displayed on the screen. Without the presentation mechanism model, the connection between pictures on the screen and underlying mechanism objects has to be maintained by program modules. In short, the issue with or without the presentation model reflects the *declarative* versus *procedural* ways of data representations in software development. The declarative way of data representation makes the system more flexible, *e.g.* the first advantage (with the presentation mechanism model), illustrated above.

### 7.3.3 The architecture of EDMECH in the viewpoint of a user interface system

From the viewpoint of the user interface system the architecture of EDMECH is described in Fig.7.8. It shows that the User InterFace Manager (UIFM) for the EDMECH system physically contains the parts of UIMS, see Definition 7.6. A specific application-related part for editing conceptual mechanism structures is reflected in *action* data, which is managed by the application manager. The figure also shows the relationship of UIFM with other two system components: MPM and MOM, discussed in Section 7.3.2.
7.4 Implementation

7.4.1 Developing a general purpose UIMS system for CIMOME
The discussion in the previous section calls for a general-purpose UIMS system which can be used in developing all user-interfaces for the design tools in CIMOME. The idea is quite geared with an earlier decision made in Chapter 2, that the user interface integration dimension is to be considered in CIMOME. The discussion so far, together with the implementation strategy formulated earlier in Section 2.5 of Chapter 2, leads to the decision to develop a general purpose UIMS under the same context of our programming resources, e.g. using FORTRAN and GMS. This system is termed Min-UIMS (MUIMS for short) hereafter in this dissertation. In the following some general ideas and features of this system are outlined.

MUIMS implementation as an ABC application.
It has been found that the characteristics of an editing system are (1) the data structure is more complex while the amount of data is small and (2) the data manipulation takes place in the main memory because the editing takes place in the run time of the system. This might be true for any interactive computer system. Therefore MUIMS is implemented as an application of ABC in the sense that the information of gadgets and state transition network is represented as a database model. This implies that the information of gadgets, state-net and actions is stored in databases. A database schema is independent of applications, while
database instances are application-specific. The gadget manager, application manager and executive manager include some generic routines (of ABC) that operate on a user-interface database, and thus they are independent of applications. In this way the development of a user interface of applications is equivalent to the instantiation of a user interface database. Two advantages are obtained: the first is the separation of an application and its interface, and the second is that such a UIMS is generic to all applications.

Figure 7.9. A User-Interface Database Model Schema: Conceptual View

User Interface database model in MUIMS
A database schema in MUIMS has been defined as shown in Fig.7.9. This model schema is developed in terms of the analogy (as shown in Fig.7.10) between the machine structure and the user-interface structure. Based on this analogy, together with the idea used to define GMM in Chapter 5, this user-interface model schema can be readily derived.

MUIMS taking shared-control mode
There are three control modes in UIMSs (see Fig.7.11). A so-called internal-control UIMS takes the form of a procedure library including user interface tools. These are activated under the control of the application program. A so-called external control UIMS assumes a total responsibility for the interface to call the application procedures to perform application-oriented tasks. Hence the control resides in the user interface, not in the application. A more advanced control mode is the so called shared-control mode. In this mode, both the application and the user-interface are allowed to control the dialogue sequence. MUIMS supports the shared-control mode. This is necessary because for complex engineering applications (e.g. machine design), the total semantics of user-computer interactions can most unlikely be represented solely in the user-interface database.

Application manager in MUIMS
The separated two parts, UIMS and application action routines, must anyway communicate.
Section 7.4

Implementation

<table>
<thead>
<tr>
<th>machine structure</th>
<th>user interface structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>machine consists of assembly &amp; part</td>
<td>user-interface consists of gadgets &amp; widgets</td>
</tr>
<tr>
<td>an assembly fulfils a function</td>
<td>a gadget fulfils a function</td>
</tr>
<tr>
<td>an assembly consists of parts</td>
<td>a gadget consists of widgets</td>
</tr>
<tr>
<td>a part can be assembled if it</td>
<td>a widget can be a gadget if it</td>
</tr>
<tr>
<td>consists of primitive parts</td>
<td>consists of widgets</td>
</tr>
<tr>
<td>assembly &amp; parts are represented</td>
<td>gadget &amp; widgets are displayed</td>
</tr>
<tr>
<td>as solid entity and perceptible and</td>
<td>as window and perceptible and operated</td>
</tr>
<tr>
<td>operated</td>
<td></td>
</tr>
<tr>
<td>assembly &amp; parts are related</td>
<td>gadget &amp; widgets are related together</td>
</tr>
<tr>
<td>together to fulfil a common goal</td>
<td>to fulfil a common goal</td>
</tr>
</tbody>
</table>

Figure 7.10. Analogy between Machine and User Interface

![Internal Control](image)

![External Control](image)

![Shared Control](image)

Legend: ○ procedure; □ UIMS; ■ application.

Figure 7.11. Control Modes of UIMS (Koivunen & Mantyla, 1988)

This communication is enacted via the application manager. The communication has several patterns, dependent on types of the information to be transferred between UIMS and applications. For example, a simple pattern might be that UIMS triggers an application action routine and the application action routine reports to the UIMS when its process is finished. A more complex pattern might be that a triggered action routine also reports which text items need to be displayed in a window.

Fig.7.12 shows an example in EDMECH with several screen copies of the real system. In window-1 if an end-user chooses boxes as shown (he wants to specify the input motion in equal steps and constant velocity). The next window, window-2, shows a set of text entry boxes for the system to get values from the end-user. However, if an end-user chooses the input motion with non-equal steps, see window-3, the next window, window-4, shows several
Figure 7.12. The Need of Shared Control Model in MUIMS
different text entry boxes, e.g. for a file name, in which displacement, velocity and acceleration are given in discrete points. In the user-interface database, it is only specified that after an input motion representation type is chosen, the next step is to get input motion values. While the semantics is left to applications, that there are two different situations which further render to display different text entry boxes. So in this case the corresponding application action routine has to return different text entry boxes to UIMS for further processing. The action database model represents those various communication patterns managed by the application manager. Since at present both the MUIMS and the applications are written in the same programming language, FORTRAN, the application manager with the action database is simply implemented as several communication pattern routines. The communication pattern routine takes a general form as such:

CASE 'action_d' =
    create_a_bar: trigger the subroutine to create instances for certain classes, e.g. BAR, MECH-BODY, etc. (see Chapter 5);
    delete_a_bar: trigger the subroutine to delete instances from certain classes;

END_CASE

(a) Atom Level (b) Molecule Level (c) Library Level

\textit{Figure 7.13. Three-Level Scheme for Editing Mechanisms}

7.4.2 The effort to reduce the number of operations
To increase the editing speed of a mechanism by experienced users, a three-level scheme atom/molecule/library is considered in the implementation. This is illustrated with an example in Fig.7.13. In the atom scheme, four machine parts are processed individually (the operation sequence for creation follows: Edit -> Create -> Body -> Select a part type from the Body window (assume \textit{bar}) -> Select an appropriate bar presentation). In the molecule scheme, a group of machine parts is processed once (the operation sequence for creation follows: Edit -> Create -> Group -> Select an appropriate group). In the library scheme a complete \textit{kinematic chain} is created once (the operation sequence for creation follows: Edit -> Create -> Library -> Select an appropriate type of kinematic chain). To enable editing at the molecule scheme, more knowledge about mechanism structure analysis is needed such as \textit{Assure}-

157
Group theory. The three-level of editing schemes shows a feature of EDMECH system, that is, a broad spectrum of end-users is accommodated, see Section 7.2.

7.4.3 The effort to support different views of the same object
Support for different views of the same object is considered as one of aspects to raise the system intelligence, see Section 5.3 of Chapter 5. In the context of editing mechanisms, the phrase different views of the same object means that end-users may have different fonds of graphic representations on the same application semantics (see Fig.7.3 and Fig.7.14a) and may have different procedures to construct the same application semantics (see Fig.7.14b).

At the implementation level the handling of the two cases shown in Fig.7.14a,b is different. In the case of Fig.7.14a, either of the two views about the frame is stored in the database. It may be possible that both views are stored redundantly. There must be a process at the implementation level to maintain this redundancy. In the case of Fig.7.14b, the different procedures are taken care by the program process, while in the database only one application semantics is described and stored, i.e. two machine parts are connected via a revolute joint. It is observed from the implementation of supporting these different procedures to create one same application semantics that it is generally difficult to specify these semantics in the user-interface database and a shared control mode of a UIMS is therefore envisaged.

(a) Views of Frame
Application semantics: mechanism frame

view 1

view 2

(b) Views of Procedures to Create a Joint

Application semantics:

- bar-1 and bar-2 are connected with revolute joint

View 2:
Create two bars first and then join

View 3:
Create bar 1 -> create joint -> create bar 2 -> complete the joint

Figure 7.14. Examples of Different Views of the Same Object in Editing Mechanisms

7.5 Discussion and Conclusions

No matter top-down and bottom-up design processes, a kind of system (in the present study, the EDMECH system) is needed which helps an end-user to specify mechanism structures and design intents. For example in a top-down design process, after a design solution has been
found by a design tool (computer program) and is displayed on the screen, there might be a need to add more bodies on the solution. In contrast to a lot of other work in which specification of mechanism structures with design intents depends on the kinematic and dynamic analysis purpose (some discussions on this point has been done in Chapter 5), EDMECH takes GMM as its underlying object model. So EDMECH is a neutral editing system to be used for many design tools in CIMOME (In Chapter 2, this point has been implied in the discussion of Section 2.4.3, and in Chapter 9 several implementations relevant to EDMECH will be further shown).

The end-user's mental model in EDMECH assumes minimal domain knowledge in the sense that this knowledge is primary to mechanical engineers with three or four year study programs. The three-level of editing scheme, in the EDMECH system, is a unique feature in CAD/CAM systems for the machine design. In the procedure to create mechanism objects, EDMECH reaches a highly flexible situation due to the shared control model of its underlying UIMS. The elaboration and classification of editor systems against CAD/CAM environments are seldom discussed in the literature.

In the following a demonstration of two sample mechanisms via EDMECH is presented. The first example concerns a bar-gear compound mechanism with an input motion being defined on body-1 with respect to a reference line as shown. Three major steps to edit this mechanism are shown in Fig.7.15. Window-1 shows that the editing starts with the library editing level by picking a four bar chain. This four bar chain is further configured using the Move button (Window-2). Window-3 shows that gear bodies are created in the molecule editing level by picking a pair of gears. This example will be further used as an illustration in Chapter 8 to see how the finite element model of this mechanism can be automatically generated. The second example is a bar-cam compound mechanism (Fig.7.16). With the background of the first example, the editing procedure for this example is not difficult to be imagined, details are thus omitted. The profile of cam body is unspecified at this moment and will be calculated during another design stage. (Chapter 9 will introduce a system called CAMDES which can calculate the cam profile.)
Figure 7.15. A Bar-gear Compound Mechanism Edited with EDMECH system
Figure 7.16. A Bar-cam Compound Mechanism Edited with EDMECH system
Automating Finite Element Modelling of Mechanisms

8.1 Introduction

GMM, which has been described in Chapter 5 concerning both its underlying philosophy and its conceptual data representation, should prove its usefulness in CIMOME for design activities at the conceptual design stage. This will be demonstrated in this chapter by exemplifying that a finite element view of mechanisms for certain design tools can be automatically generated from GMM - in a more broad sense the idea described in Chapter 4, that views can be derived from their underlying objects. Choice of the finite element view is made because some important members to be brought into CIMOME are based on this view, e.g. RUNMEC and OPLAM (see Section 2.2.3 and Section 2.4.1 of Chapter 2 for their definitions). GMM, however, does not exclude some other well known views (e.g. a bond graph view of mechanisms).

At the aspect of kinematic and dynamic analysis or simulation of mechanisms the automatic generation of the finite element view implies that knowledge of FEM, which underlies tools such as RUNMEC, can be completely hidden from the end-user. This, together with an earlier achievement - the EDMECH system (Chapter 7), leads to a highly intelligent computer-support paradigm in which an end-user merely draws a sketch of the kinematic mechanism diagram with EDMECH so as to simulate the motion behaviour of the mechanism. (The paradigm has been predicated earlier in Section 5.2 of Chapter 5 with Fig.5.5.) It is noted that such a paradigm has not yet been reached in the research domain of CAD/CAM of machines, as commented by Tsai et al. (1992): however, the application of such general-purpose tools as an integral part of the design process has been limited by factors inherent in their design. A part of the problem is that such tools have been developed primarily by and for specialists. Other limiting factors include the level of detail typically needed for input
specifications.

At the aspect of dimension synthesis the automatic generation of the finite element view predetermines another highly intelligent computer support paradigm for dimension synthesis via optimization. This is so because on one hand a tool called OPLAM is based on the finite element view and on the other hand the OPLAM system is the most general system in comparison with those found in the literature, e.g. those developed by Kramer & Barrow (1990) and by Sodhi (1988).

The general approach to building such a system (the system is hereafter called GMMFEM meant for 'from GMM to FEM') is model-based (Kunz et al., 1989). The model-based approach can be characterized with several points: (1) a domain object structure model, i.e. GMM in our case, (2) a management system for the domain object model, i.e. Mechanism Object Manager - a program management system upon GMM and (3) a production system which plays a role as a kind of expert for the finite element modelling of mechanisms.

The organization of this chapter is as follows. In Section 8.2 review and classification of the finite element method are presented. (Since this finite element method was originally developed at Delft University of Technology, hereafter it is called Delft-FEM for short.) The complexity of the automatic generation of the finite element view is also exhibited. In Section 8.3 major concepts to build this system are elaborated. The discussion shows the need to develop a so called finite element mechanism model (FEMM for short) - a data representation of evolutionary modelling results with FEM. In Section 8.4 a conceptual schema of FEMM is presented for which the discussion in Section 8.2 serves application semantics. In Section 8.5 reasoning strategy and reasoning are discussed. In Section 8.6, examples are demonstrated in a sub-environment called KINANA for the kinematic simulation of mechanisms. KINANA contains or references to EDMECH, GMMFEM and RUNMEC. (Definition of KINANA can also be found in Section 2.4.1 of Chapter 2.) Through the discussion of these examples, the advantage of the system can be identified. Discussion and conclusions are given in Section 8.7.

8.2 Delft-FEM: Review and Classification

8.2.1 Review

With the objective to establish the constrained motion equation of the mechanism, Delft-FEM was first described by Van der Werff (1975-1977). The very starting point is to view a mechanism as a set of finite elements and as such (1) the topology and geometry of a mechanism is replaced (viewed) by those of finite elements and (2) the continuity equation, as well known to the general finite element theory, corresponds to the constrained motion equation of the mechanism. The equation for the dynamic behaviour of a mechanism can
similarly be established by the corresponding finite element fundamental equations such as the constitutive equation and force equilibrium equation usually represented by the virtual work principle.

![A mechanism](image1)

![A finite element view of mechanism](image2)

**Figure 8.1. A Finite Element View of a Mechanism**

**The core concept of Delft-FEM**

In terms of the general finite element theory the configuration of a mechanism in a certain position is determined by a set of node coordinates of all elements measured from a global coordinate system $X_X-Y_X$. The set can be understood as a vector; $x$ denotes the vector of coordinates of nodes, $x = [x_1, x_2, \ldots, x_n]^T$. Among $x$, a subset of the node coordinates are known, for example the coordinates of the nodes $A$ and $D$ (Fig.8.1). Let $x^o$ denote the known coordinates and $x^c$ denote the unknown coordinates, and thus $x = [x^o, x^c]^T$. Elements are subject to deformations which are denoted by $\varepsilon$ - a vector of deformations of each element, i.e. $\varepsilon = [\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n]^T$ (h denotes the total number of deformations in a system. Similar to $x$, deformations can be classified into knowns ($\varepsilon^o$) and unknowns ($\varepsilon^c$), thus $\varepsilon = [\varepsilon^o, \varepsilon^c]^T$.

The kinematic problem is equivalent to finding $x^c$ for prescribed $\varepsilon^o$. The $x^c$ is governed by a system of mathematic equations called constrained motion equation. To formulate this equation, a set of finite elements with certain types is applied to a particular mechanism structure. Types of elements should be defined which relate $x$ of that element to kinematic parameters, such as length of a bar. Because the continuity equation in the general finite element method specifies the relationship between $x$ and $\varepsilon$, deformations of elements should be defined relevant to the kinematic parameters. Fig.8.2 shows three major types of elements which are sufficient for the kinematic and dynamic analysis (rigid and flexible) of most planar bar mechanisms. (Below the term form parameter refers to the prescribed value of the parameter such as $f$ in Fig.8.2.) For each type of element, a system of equations is defined which is the continuity equation: $e_{ik} = g_{ik}(x_{ik})$, where $k$ denotes a specific element type. For

165
element type beam(I) (Fig. 8.2b), three equations are defined below:

\[
x = \begin{pmatrix} x_p, y_p, \beta_p, x_q, y_q, \beta_q \end{pmatrix}^T
\]

--- definition of coordinates

\[
\varepsilon = \begin{pmatrix} \varepsilon_{1}, \varepsilon_{2}, \varepsilon_{3} \end{pmatrix}^T
\]

--- definition of deformation modes

\[
\varepsilon_{1} = \sqrt{(x_q - x_p)^2 + (y_q - y_p)^2} - \ell
\]

--- 1st continuity equation

\[
\varepsilon_{2} = [\beta_p - (\beta - \beta')] \ell
\]

--- 2nd continuity equation

\[
\varepsilon_{3} = [(\beta - \beta') - \beta_q] \ell
\]

--- 3rd continuity equation

Figure 8.2. Three Fundamental Elements

In accordance with the general finite element theory, definition of element types should ensure the mutual compatibility of displacements on the element boundaries (Zienkiewicz, 1977). In the case of mechanism structures this can be achieved by simply sharing coordinates at nodes among mutually bounded elements. The coordinates include not only the 'normal' coordinates like the translating movement \((x, y)\) but also the orientation movement, for example \(\beta_p\) and \(\beta_q\) of planar beam elements in Fig. 8.2 (generalized coordinates). In the following, terms such as translating coordinate and orientation coordinate will be used. A translating node contains a set of translating coordinates and an orientation node contains a set of orientation coordinates. Care should be taken of \(\beta_p\) and \(\beta_q\) which are defined as being measured from the reference position. The reference position is usually the initial position of a mechanism. This treatment ensures the boundary compatibility of several rigidly joined elements by sharing coordinates, irrespective of coordinate systems (see Fig. 8.1b, three beam elements share one orientation coordinate \(\beta_f\) at the point F). By assigning elements to machine bodies and assembling all continuity equations of elements, the system continuity equation (a set of non-linear equations) can be obtained:

\[
\varepsilon = \varepsilon(x)
\]  

(8.1)

For the kinematic analysis of the rigid body system displacement due to the prescribed deformation of the bodies is neglected in comparison with the movement of a body. Therefore parts of \(\varepsilon_i\) \((i = 1, 2, \ldots, h)\) have zero deformation; eq.(8.1) yields:
\( g_i(x^e) = 0, \quad i=1,2,\ldots,w, \; w \leq h. \) \hspace{1cm} (8.2)

The equation above does not consider the degrees of freedom (input motions). In Delft-FEM, input motions of mechanisms are modelled by designating coordinates as well as deformations with a prescribed value, as shown in Fig. 8.3. Therefore both \( x \) and \( \varepsilon \) can be further classified into \( x = [x^T, x^{mT}]^T \) and \( \varepsilon = [\varepsilon^T, \varepsilon^{mT}]^T. \) The superscript 'm' denotes that the corresponding \( x \) and/or \( \varepsilon \) shall perform input motions or is time-dependent. With this classification, eq. (8.2) can be further solved in a computable scheme; for details see (Van der Werff, 1977).

![Figure 8.3. Finite Element View of Input Motion](image)

The deformations and coordinates in the definition of three major types of elements (Fig. 8.2) have a one-to-one relation with physically meaningful strains and external forces, respectively. Therefore the constrained motion equation due to external forces can be derived by following the standard procedure based upon the virtual work principle (Zienkiewicz, 1977); for details, see (Van der Werff, 1977).

**The extension of the Delft-FEM**

Extensions to the core concepts of FEM are performed along two routes (i) towards the applicability for gear, cam and high pair mechanisms and (ii) towards easy modelling for kinematics-oriented design activities, such as the synthesis through kinematic optimization (Klein Breteler, 1987). As a consequence, additional types of elements have been defined. Parts of new element types are shown in Fig. 8.4. (For convenience of the following discussion, an identifier for each element type is given.) Some important features of these new types of elements are discussed below:
**Beam (II) element**
coordinate: $x = |x_p y_p \beta_p x_q y_q \beta_q |^T$
form parameter: $\varepsilon = |\ell, \psi_p, \psi_q |^T$

**Pivox (II) element**
coordinate: $x = |\beta_p \beta_q |^T$
form parameter: $\varepsilon = |\beta_{pq}|$

**Binary element**
coordinate: $x = |x_p y_p x_q y_q |^T$
form parameter: $\varepsilon = |\ell, \beta|$

**Ternary(I) element**
coordinate: $x = |x_p y_p \beta x_q y_q |^T$
form parameter: $\varepsilon = |u, v, \beta |^T$

**Pclamp element**
coordinate: $x = |x_p y_p \beta x_q y_q |^T$
form parameter: $\varepsilon = |\ell, \beta |^T$

**Gear(I) element**
coordinate: $x = |\beta_a \beta_b \beta_c i |^T$
form parameter: $\varepsilon = |g|$, where $g=i(\beta_a-\beta_b)-((\beta_c-\beta_b)$

**Half beam element**
coordinate: $x = |x_p y_p \beta |^T$
form parameter: $\varepsilon = |\beta |$

cordinate: $x$=any type mentioned above
form parameter: $\varepsilon = |x |$

*Figure 8.4. Several Additional Elements Supported by RUNMEC*
The overall orientation $\beta$ of elements was introduced as a coordinate in some elements (when this coordinate can be varied independently), see the definition of element types (P10 and P7). This forms a basis for a variety of element types, e.g. the disk-cam element (P8). Angle $\beta$ is measured from the x-axis of the global coordinate system, which differs from $\beta_p$ and $\beta_q$ in the original definition of the Beam (I) type (Fig.8.2).

The overall orientation $\beta$ of elements must be viewed as a form parameter when $\beta$ is determined by the coordinates of that element; see element types (P7 and P10). The binary element (P6) can be considered as an extension to the truss element (P1) to facilitate the modelling of body structures, with regard to the kinematic design, see Fig.8.5a.

The point-oriented orientation coordinate $\beta_p$ is defined with respect to the global coordinate system (see definition of element type Beam (II) (P4) in Fig.8.4), instead of the reference position (see definition of element type Beam (I) (P2) in Fig.8.2). This is intended to be consistent with definition of the $\beta$ coordinate in the element types (P7 and P10). Care must be taken that it is no longer possible to make three beam elements with type Beam (II), for the structure shown in Fig.8.5b, boundary compatible at point F by sharing the orientation coordinate, as it does with three beam elements with type Beam (II), see Fig.8.1. A new element type Pivot (II) (P5) should be defined to make the boundary compatible in the case of three Beam (II) elements. The difference of element types (P5) and (P3) is that in element type (P3) two orientation coordinates are defined with respect to the reference position, while in element type (P5) they are defined with respect to the global coordinate system.

The displacement compatibility on the boundary of elements is normally ensured by sharing boundary nodes of each element, which may be called explicit sharing. In the kinematic analysis of mechanisms, however, a technique which would be called implicit sharing is useful. By implicit sharing is meant that node coordinates of two elements are subject to a time-independent protocol among deformations of these corresponding elements. In practice, the time-independent protocol can be:

$$\varepsilon_{1(0)} - \varepsilon_{2(0)} = \text{constant}, \quad \text{or} \quad \varepsilon_{1(0)} + \varepsilon_{2(0)} = \text{constant}$$

(see Fig.8.5c,d),

where $k, j$ denotes different elements and 1,2 denotes $i$-th deformation in an element. The implicit sharing of boundary nodes can also be viewed to construct a compound form parameter.

A special element type called unit element (P11) was introduced to convert a coordinate into a deformation. This is for the purpose of kinematic optimization of mechanisms, where only form parameters are considered as design variables of the objective function (Klein Breteler, 1987).

### 8.2.2 Classification versus separation of kinematics and dynamics tasks

In terms of whether the orientation coordinate is defined globally (the angle is measured with respect to the global coordinate system) or locally (the angle is measured with respect to the
Body DC is modeled by a binary element form parameter ($\beta$) is prescribed with the value $\alpha$.

The structure below is modeled by three Beam (II) elements. Two Pivot (II) elements are needed to make the boundary compatibility of three beams.

A typical structure

Beam (II) element
Pivot (II) element
element type P4

Figure 8.5. Guidelines for Finite Element Modelling of Typical Structures

reference position of a body), element types are classified here into two kinds; the former is called global-based and the latter local-based. For example all the element types shown in Fig. 8.4 are global-based. Several local-based element types can also be found in (Kooiman, 1987), besides these shown in Fig. 8.2. The local-based element type is, in general, rooted in the structural analysis in the field of technical mechanics, where the mechanism motion
problem is viewed as the variation of a structure from its initial position. The global-based element type is, in general, rooted in the kinematic analysis of mechanisms in the field of mechanisms, where the mechanism motion problem is viewed as the relationship between two bodies (one body might be the global frame and thus the motion is measured with respect to the global coordinate system). The global-based element type is more suitable for the kinematic analysis problem. This is mainly because the initial position of a mechanism is not required to be given exactly, e.g. in the program system RUNMEC (notice that calculating precise initial positions of a mechanism is equivalent to doing kinematic analysis itself).

The discussion above implies that separation of kinematic and dynamic analysis tasks should be considered, as mentioned earlier in Section 2.4.1 of Chapter 2. Global-based element types would be more suitable for kinematic analysis while local-based elements (e.g. Beam (I)) would for the dynamic analysis\(^1\). It is noted that this separation also makes sense in reducing the complexity of finite element modelling. For example, for the purpose of dynamic analysis, assuming the initial position of a mechanism known, using three Beam (II) elements to model the structure BCFE shown in Fig.8.1a will be more complex than using three Beam (I) elements. (The initial position can be obtained with first performing the kinematic analysis.) This conclusion can be derived from the earlier discussion about the difference between Beam (I) and Beam (II). The finite element modelling for dynamic simulation needs also to consider material distribution; thus reduction of the complexity at the aspect of the modelling for mechanism structures will be helpful to the automatic finite element modelling for the dynamic analysis or simulation.

### 8.2.3 A modelling case with RUNMEC

Fig.8.6a shows a bar linkage with three degrees of freedom. The finite element model of the FEM-based analysis package RUNMEC is graphically shown in Fig.8.6b. The input data file of this model is shown in table-8.1 to give an impression. Several remarks regarding this model are illustrated below (more details about the format of this file are referred to Nowe (1993)):

- Six binary elements are used for the purpose of kinematic analysis.
- Each element actually implies its direction with the sequence of coordinates. For example element nr.2 has a direction from node B to C, and element nr.3 from node C to D. Nodes A,B,C and D are translating nodes and have coordinates labelled as shown in Fig.8.6b.
- Labelling of coordinates follows a rule: first to label fixed nodes and then moving nodes.
- The element boundary compatibility between element nr.2 and element nr.3 is via the

---

\(^1\) Zhang & Klein Breteler (1990b) observed that for the kinematic analysis, the element type Beam (I), which is remarkable with their point-oriented orientation coordinates, can be replaced by the Binary element type. Following this route, Zhang & Klein Breteler (1990a) developed types of elements for the spatial mechanism analysis. More detailed discussions are out of the scope of the present study.
*implicit sharing* as illustrated in the last section, i.e. $\beta_3-\beta_2=0.0$ ($\beta$ is the form parameter of the binary element, see Fig. 8.4). Care should be taken of the direction of corresponding elements nr.2 and nr.3. For instance, if the direction of element nr.2 is from node C to B, the constrained equation for the element boundary compatibility should be $\beta_3-\beta_2=\pi$. The technique to represent it in the model is via multiplication of a factor 1.0 to the form parameter $\beta$ of element nr.3 while of a factor -1.0 to that of element nr.2.

- The elongation deformation of element nr.3 should be released. This is represented in the model such that the elongation deformation (marked by keyword LENL) of element nr.3 is not listed under the keyword FORM.

- The time-dependent prescribed motion (or input motion) is represented by form parameters of some binary elements in the model. (It is noticed that in RUNMEC the input motion cannot be specified with coordinates.) Such form parameters must be labelled as last numbers, see form parameters 7, 8, 9 under the keyword FORM.

![Figure 8.6. A Case Study of the Finite Element Modelling](image-url)

This simple example has provided an evidence of the comment made by Tsai *et al.* (1992), mentioned earlier in Section 8.1 of this chapter, that is, the system requires too detailed levels of input specification for non-specialists and is designed for specialists. Facilitating the finite element modelling of mechanisms is therefore needed. The following discussion will be around the issue of the automatic generation of this model.

**Table-8.1. The input data file of RUNMEC for the case shown in Fig.8.6a**

<table>
<thead>
<tr>
<th>KIN ANA</th>
<th>; kinematic analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TOPOLOGY</strong></td>
<td></td>
</tr>
<tr>
<td><strong>ELEM</strong></td>
<td><strong>BIN</strong></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>11</td>
</tr>
</tbody>
</table>

172
The General Solution to Automating Finite Element Modelling

8.3 The General Solution to Automating Finite Element Modelling

8.3.1 Requirement analysis
The context of the system to perform finite element modelling, i.e. GMMFEM system, is a machine motion analysis or simulation sub-environment of CIMOME, the KINANA system. KINANA is organized into two levels: the user level and the computer level, as implied earlier in Chapter 5, Fig.5.5b. The machine objects with designers' intents are entered into the computer system with an editor system EDMECH (see Chapter 7) and are further represented in GMM (see Chapter 5). The GMMFEM system corresponds to a process c in Fig.5.5b of Chapter 5. The activity of the GMMFEM system is no more than to analyze GMM so as to generate a model for individual FEM-based packages such as RUNMEC. This model is usually represented as an input data file. In the KINANA environment the results (x and/or ε) of the calculation with RUNMEC are further, through a system called FEMGMM, related to certain machine objects in GMM and presented to the end-user with those known to the end-user (assuming that the end-user does not know the finite element method at all). The FEMGMM system corresponds to a process c' in Fig.5.5b of Chapter 5.
A simple example is discussed to give an impression of the tasks of GMMFEM and FEMGMM, respectively. In Fig.8.7 a four-bar linkage is shown with input motion $\alpha$ and output motion $\beta$. In GMM both input and output motions are specified in accordance with the user-oriented reference system (for definition, see Section 5.5.3 of Chapter 5) - in the current case a reference line on the ground. GMMFEM must convert this user-oriented input specification ($\alpha_{\text{user}}$) into time-dependent prescribed orientation deformation ($\alpha_{\text{FEM}}$) of the binary element representing body-1. While FEMGMM converts the output (resulting from RUNMEC) which is represented as an orientation deformation ($\beta_{\text{FEM}}$) of the binary element representing body-3 to the user-oriented output ($\beta_{\text{user}}$). (The current version of RUNMEC (Nowe, 1993; Hamoen, 1994) does not allow calculated deformations to be specified as output; assigning a half beam element, element type (P10), to body-3 for that purpose is generally needed.) This simple example manifests a scheme for complete hiding of FEM for the end-user.

![Diagram of a four-bar linkage](image)

**Figure 8.7. The User-oriented Reference and the FEM-based Reference**

8.3.2 Concepts for solution

The discussion in the following will focus on the establishment of the GMMFEM system. From the aspect of the task of GMMFEM, the system must be an expert system - a finite element modelling expert. It is observed that the so called model-based reasoning approach (Kunz et al., 1989) is very much suitable for the problem of finite element modelling. The model-based approach in the present study is defined as: (1) the structure of a domain and the principles which characterize its behaviour are explicitly represented and (2) the representation method is of semantic data model. The model-based approach differs from the classical expert systems in that they only represent empirical associations as described by experienced human experts. Some reasons for the model-based approach to automating finite element modelling are:

- The knowledge of the finite element modelling is more about structure than empirical associations. For example, a binary element has two translating nodes (or four translating coordinates) and two deformation modes (see Fig.8.4). The principle to define the binary element (actually for any type of elements) is that the number of prescribed deformations is equal to the number of coordinates diminished by the number of degrees of freedom of
the element as a rigid body (Van der Werff, 1977). It will be difficult to represent such knowledge with IF-THEN formalism of classical expert systems.

- The result of the finite element modelling itself is a kind of view of a mechanism in terms of the finite element formalism, as described in Section 8.2 (especially, see Fig.8.6b and table-8.1 about a case study). The representation of this view is not just a matter of pieces of statements for a certain conclusion, which is frequently seen in classical expert systems, e.g. MYCIN system.

- In a preliminary study of the possible modelling procedure with the computer, each body of a mechanism is in turn evaluated against its type and its connection features with other bodies and then appropriate types of elements can be assigned. This procedure, however, gives rise to a problem that the decision at one step of reasoning for one body may have to be postponed due to lack of information. Consider the modelling of structure BFCE of Fig.8.1 as an example. Assume that body-7 is modelled, see Fig.8.8a, and that two binary elements are assigned, see Fig.8.8b. However, the issue that body-7 has a fixed joint with body-9 at point F cannot be modelled so far because body-9 has not yet been modelled (none of element is assigned on it). The modelling decision on body-7 has to be postponed until body-9 is modelled. The postponed information must be marked in the system such as: 'body-7 is not completely modelled and the unresolved issue is the connection of body-7 with body-9.' Obviously such kind of information is about structure of a domain rather than empirical associations.

![Diagram](image)

*Figure 8.8. A Case of Postponed Modelling*

The analysis presented above implies that a kind of (application data) model is needed which explicitly represents:

1. knowledge about types of finite elements described in Section 8.2,
2. information about correspondence between bodies and finite elements which have been assigned to bodies,
3. relationships (or constraints) among a group of assigned finite elements and geometry of these finite elements, and
4. time-dependent prescribed coordinates and/or deformations corresponding to the input motions.
Hereafter this model is called Finite Element Mechanism Model (FEMM). The representation of this model will utilize the data modelling method described in Chapter 3.

It is also recognized that in finite element modelling, a kind of knowledge, featured as associations, is needed as well. (Definition of knowledge can be found in Appendix A.) Such kind of knowledge is better represented with the IF-THEN formalism. So in general GMMFEM system contains two types of knowledge with their preferred representation methods.

FEMM stays between GMM and an input data file of a specific FEM-package. This provides two extra opportunities of the KINANA system:

- There are two kinds of end-users: one with no or little knowledge of FEM (unexperienced user) and the other with good knowledge of FEM (experienced user). With the explicit representation of the finite element modelling result available, the experienced user can browse the modelling result and do some modifications (delete, create and update elements on mechanism structures) on the one created by the computer system. This might be more significant in the finite element modelling for dynamic simulation or analysis. Still this kind of end-users is not necessarily involved in the level of detailed input specifications for a specific package.
- In the analysis of a large mechanical system a fast response is expected when only a small change of geometry of the system is made (Tsai et al., 1992). The FEMM will be very helpful in this respect. Suppose that the length of a bar FC (Fig.8.1) is changed from 100 to 200 only the geometry of two elements on this body is affected. Adaption of this change can be performed at the FEMM level. The complete re-modelling of the whole mechanism is then avoided.

8.3.3. GMMFEM system representation
Fig.8.9 presents the GMMFEM system with the data flow diagram (Hawryszkiewycz, 1988). The system shown in this figure reflects the thoughts discussed in the last section. Several points need to be illustrated further.

- FEMM is divided into a generic part and a specialized part. The generic part includes the knowledge common to different FEM-based packages, e.g. SAM (ARTAS, 1994), RUNMEC and SPACAR, etc. The specialized part includes the knowledge only used in a specific FEM-package for its own purposes. Several evidences can be exhibited to help clarify this point.

Case (1) (see Fig.8.10a).
Semantics at user-side: Node A needs to be varied to examine its influence on an output path. In this case the two coordinates of node A belong to the design variables.
Case (2) (see Fig.8.10b).
Semantics at user-side: The transmission ratio $i_{12}$ of a pair of gears needs to be varied.
Generic description with FEM: Define the transmission ratio $i_{12}$ as a kind of coordinate with element type (P9), see Fig.8.4, and this coordinate must be varied.
Specialized description with OPLAM: Define a unit element corresponding to this coordinate to convert this coordinate into a form parameter for variation.

Case (3) (again see Fig.8.10a).
Semantics at user-side: Node A is fixed.
Generic description with FEM: Translating node A is fixed at both x direction and y direction.
Specialized description with RUNMEC: Label x and y coordinates of node A with smallest number, also see the case example described in Section 8.2.3.

The whole process (c) is further divided into processes c1 and c2. Process c1 takes the data resource from GMM and knowledge base (A). This knowledge base contains the knowledge which makes more sense of association; for example a piece of knowledge would be: IF the modelling task is for the kinematic analysis THEN only the binary
element type is applied. More discussions about such kind of knowledge can be found later in Section 8.4. The data generated from process $cl$ is an instantiated FEMM (generic part). Process $c2$ takes the resource from FEMM and knowledge base (B). Knowledge base (B) contains rules specific to individual FEMM-packages. Process $c2$ generates both the data instantiated to FEMM (specialized part) and an input data file for individual packages.

Both processes $cl$ and $c2$ have a two-way data flow related to FEMM. This is so because the reasoning process is such that the preceding conclusions (decision about the element assignment on machine bodies) might become the condition for the subsequent reasoning. Fig. 8.11 exhibits the modelling progress in the viewpoint of the instantiation of FEMM.

![Diagram](image)

Figure 8.10. Generic Finite Element Method vs Special Treatments

![Diagram](image)

Figure 8.11. The Modelling Process with Respect to the Instantiation of FEMM

8.4 A Finite Element Mechanism Model

The conceptual schema of FEMM will be presented with two parts. The first part is the data representation about the knowledge of each element type. The second part is the data representation of the modelling result after assigning elements with certain types on a
Section 8.4

A Finite Element Mechanism Model

particular mechanism; such a modelling result is like a snapshot of the picture of Fig. 8.1b. The information in the first part can be thought of as meta information of the second part. The definition of the schema in the first part is therefore inspired from guideline-1 of the conceptual data modelling described in Section 3.3.5 of Chapter 3.

8.4.1 Data representation of the element type definition

According to the discussion on the knowledge of the element type (Section 8.2) a model is presented in Fig. 8.12 with two defined classes, where possible instances considered at the implementation level are also shown. The attribute 'deformation-number' corresponds to the arrangement of deformation modes defined in a particular element type. For example in the definition of element type Beam (I) (see Fig. 8.2) the arrangement of deformation modes is such that the first deformation mode corresponds to elongation along the axis connected to two end points. It should be noted that the information about node types of each element type is not represented in Fig. 8.12 and will be discussed in Section 8.4.2.

**Figure 8.12. Data Representation of Finite Element Types**

From Section 8.2, it becomes clear that node in the finite element theory is no longer just a normal node which contains x, y and z coordinates. This is also true for coordinates. (In the following, terms the FE-node and the FE-coordinate will be used.)

A FE-node contains a number of FE-coordinates with possibly different types of FE-coordinates. Fig. 8.13 presents the model for the semantics. It can be seen that the underlying philosophy of this model is similar to the model shown in Fig. 8.12. In particular, the attribute 'coordinate-number' corresponds to the arrangement of coordinates defined in a particular FE-node type.
8.4.2 Data representation of the FE-view of mechanisms

The FE-view of mechanisms is formed when individual elements of different types are assigned to mechanism structure. To cover the geometrical and topological semantics of an original mechanism, both the elements and the relationships between elements with respect to the mechanism must be represented. The information about elements includes both element structure (geometry, mass, place, etc) and element behaviour (deformation and coordinate characteristics: prescribed or calculable). The information about the relationship between elements includes constraint between elements. The constraint reflects the boundary compatibility in the viewpoint of the finite element theory and the topology in the viewpoint of the mechanism which is modelled with finite elements.

Fig 8.14 shows a model which represents individual finite elements (with the class ELEMENT and its sub-classes) and the binding of elements with machine bodies (with the class ELEMENT-BODY). In this model, the semantics that has not yet been described in the model shown in Fig 8.12, i.e. FE-node (types and the number of nodes with a specific type) are described. For example, element type TRUSS has two translating nodes. All these FE-nodes of individual element types are classes of the type FE-NODE which will be discussed later. The instances shown in Fig 8.14 imply that both elements and FE-nodes are labelled. The boundary compatibility via sharing the node (the explicit sharing as mentioned in Section 8.2.1) can be easily implemented with both elements by taking the same node label as shown in the example of instances of Fig 8.14. With attribute NODE, class ELEMENT-ASSIGNMENT represents not only which bodies an element covers but also the precise information about the coverage. This is reasonable because in its very nature any type of the finite element is characterized with nodes. Fig 8.15 shows the instances considered for the
implementation, where a Ternary (I) element (element identifier P7, see Fig. 8.4) is assigned on a particular portion of the mechanism structure.

![Diagram showing examples of instances for ELEMENT and BINARY types with labels and mass attributes.]

**Figure 8.14. Data Representation of Finite Elements on Machine Bodies**

![Diagram showing a bar-1 and bar-2 structure with nodes A, B, C, D, and E.]

**Figure 8.15. An Example of Instances for the Relationship between Body and Element**

Fig. 8.16 shows a data model which represents individual FE-nodes and FE-coordinates. Because the node which belongs to the mechanism structure is defined as an attribute of the FE-node the complete information about where and how a particular element is assigned on machine bodies is now represented (together with the information represented in the model shown in Fig. 8.14). The precise meaning of a coordinate can be found in the class FE-
COORDINATE-TO-NODE (Fig. 8.13) in terms of attribute 'coordinate-number' of class 'FE-COORDINATE' and attribute 'type' of class FE-NODE. In this sense the model shown in Fig. 8.13 is a meta model.

![Diagram of FE-COORDINATE and FE-NODE](image)

**Note:**
Domain ('characteristic')="(time-dependent-known", 'time-independent-known', 'unknown')

**Examples of Instances:**
- For type FE-NODE:
  - <3, 'planar-translating-node', A, {1, 2}>
- For type FE-COORDINATE:
  - <1, 1, 1.5, 'time-independent known', 3>
  - <2, 2, 1.0, 'time-independent known', 3>

**Legend:**
- [ ] class from GMM
- [ ] global coordinate system
- [ ] ground point

*Figure 8.16. Data Representation of FE-Node and FE-coordinate*

Two pieces of semantics have not yet been represented above. The first is the constraint between element via implicit sharing (see Section 8.2.1 for definition). The second is the element behaviour, i.e. the characteristics of the deformations of an element when it is applied onto a particular portion of the mechanism structure. As noticed from the discussion about the implicit sharing in Section 8.2.1, the consequence of such a sharing actually produces extra prescribed deformations. So the representation of these two pieces of semantics are brought together as shown in Fig. 8.17. It shows that a prescribed deformation can be of two types: involving a single element or two elements. Because a joint element, such as element types Pivot (I) and Pivot (II), is an element in itself and at the same time it relates to other elements, it is defined as the sub-type of 'single-element' as well as the sub-type of 'two-elements'. Attribute 'joint-description', which has yet to be itemized, will include the information, e.g. whether \( e_{1(0)-e_{2(0)}} \) or \( e_{1(0)+e_{2(0)}} \) and whether \( e_{1(0)-e_{2(0)}} \) or \( e_{2(0)-e_{1(0)}} \) is meant, etc. Such information is important to specify a correct form parameter; an example is shown in Fig. 8.18.
8.5 Reasoning Strategy and Reasoning: an outline

8.5.1 Reasoning strategy

Strategy-1.
The reasoning process is generally such that each body of the mechanism is in turn evaluated against its type and its connection features with other bodies; elements with appropriate types are tailored to be assigned onto that body. While doing so the element’s structure, behaviour and its relationship with other elements are determined and recorded into FEMM. To depict the reasoning progress, for each body in the mechanism, a modelling status variable, \( s_i \) for short (\( i = \) body identifier) is defined. \( s_i \) has at least three status values: (1) body-i has yet to be
modelled, (2) body-i has been completely modelled and (3) body-i has not been fully modelled. For implementation consideration, \( s_i = 0 \) for status (1) and \( s_i = 2 \) for status (2). Status (3) may be further classified into several cases for which at the implementation level an integer number can be applied. In the following integer number '1' is used to denote this status in general.

**Strategy-2.**

For the modelling procedure (one body is considered *per se*) to work, the necessary condition is to disassemble the mechanism. The disassembly process is equivalent to extracting the structure feature of an individual body (see Fig.8.19a). The *structure feature* is generally composed of two parts: the body itself and the connection environment of the body. This leads to the definition of the *structure feature set* for each body in the mechanism, denoted by \( c_{ij} \) for short (i=body identifier, \( j=1,2,...,n \), \( n \) denotes the total number of features). Considering Fig.8.19b, \( c_{ij} \) is defined below:

- **1st feature item**, \( c_{i1} \), represents the type of body; the value of \( c_{i1} \) would be: bar, gear, rack, cam, etc.

- **2nd feature item**, \( c_{i2} \), represents a so called *non-inherent point-oriented connection* feature. The *non-inherent feature* of a body has the similar meaning as that described in Section 5.5 of Chapter 5. For the bar body-i in Fig.8.19b points A and C belong to this category. The *point-oriented connection* means the joint, *e.g.* revolute, fixed, etc. In the case of Fig.8.19b, \( c_{i2} = 2 \); this means that there are two non-inherent feature points (on body-i) through which body-i will be connected with other bodies.

- **3rd feature item**, \( c_{i3} \), represents the point-oriented connection with prescribed motions. In the case of Fig.8.19b at point B body-i has a constant prescribed point-oriented connection with body-7 and at point C body-i has two time-dependent prescribed point-oriented connections with body-3 and body-6, respectively. So in the case of Fig.8.19b, \( c_{i3} = 3 \).

- **4th feature item**, \( c_{i4} \), represents the situation that body-i is connected to other bodies with the sliding joint. In the case of Fig.8.19b body-i is connected two bodies with sliding joint, so \( c_{i4} = 2 \).

- **5th feature item**, \( c_{i5} \), represents the situation that body-i is connected to other bodies with a sliding joint which performs a time-dependent prescribed motion. In the case of Fig.8.19b, \( c_{i5} = 1 \), i.e. the relative translation between body-i and body-j is an input motion.

Sets of structure features of individual bodies may have intersections. This implies that two relevant bodies have to be simultaneously considered for modelling those intersecting parts. A protocol to decide the point of time to model two bodies simultaneously (if they have an intersecting part) is: to postpone it until the last body among two is modelled for its own right.
Figure 8.19. The Concept of Structure Features of a Body

**Strategy-3.**
Whenever it is found that body-i has a sliding connection with another body, say body-j, the reasoning process will immediately turn to finding a group of those bodies which are in series connected by sliding joints and of course initiated by body-i. Take the case of Fig.8.6 as an example. When the reasoning process reaches any one of the bodies (3,4,5) all three bodies will be retrieved. The structure feature of the sliding joint among these bodies is first modelled and then this group is just simply viewed as a ‘frozen bar’ for the further modelling. The further modelling only considers the 3rd item of the structure feature by putting all bodies in this group as a whole. The reasoning process for a single bar for the 3rd item of the structure feature can then be applied. For the case of Fig.8.6, two binary elements with one length form parameter released and the other time-dependent prescribed are assigned to this group and further modelling is not necessary because there does not exist any 3rd item of the structure feature on this group (bodies: 3,4,5).

**Strategy-4.**
In modelling of gear-bar mechanisms, the process will postpone consideration of gear bodies until all bar bodies are modelled. The underlying reason for this strategy is an observation drawn from the present study that in the gear-bar mechanism bar bodies actually build a framework while gears can be viewed as being added to this framework. Fig.8.20 exhibits this observation with an example, see Fig.7.15 (Chapter 7). This implies that modelling of bar bodies can be carried out without consideration of gear bodies provided that there is none at all any prescribed constraint between bars and gears. Under this strategy the reasoning block for the bar body in bar mechanisms can readily be reused in bar-gear mechanisms.

This strategy will also be applied to finite element modelling of the mechanism containing
machine objects such as wheel and cam bodies.

\[ a \text{ group of gears} \]

\[ \text{bar-framework} \rightarrow \text{bar-gear composite mechanism} \]

**Figure 8.20. Bars vs Gears in a Bar-gear Composite Mechanism**

**Strategy-5.**

The modelling for the rack body in a mechanism takes the following procedure. **Firstly**, a rack is viewed and modelled as a bar. **Secondly**, a rack is viewed as a gear and modelled together with its connected gear body.

**Strategy-6.**

As it has been shown from the discussion above, the scheme to automate finite element modelling is complex. In the viewpoint of rule-based reasoning, generic control of the reasoning process is a kind of forward-chain scheme with variation - a rule may directly trigger another rule. This implies that rules are structured into layers. A set of rules might be called reasoning block. The overall reasoning scheme can be depicted as shown in Fig.8.21. At level-1 the type of a body is evaluated and directed to an appropriate reasoning block on level-2. The structure features of a body are evaluated and respectively directed to individual reasoning blocks on level-3 to model a particular structure feature. An 'AND' node is employed at level-2 in the scheme diagram to express that the modelling of a body is finished only when all its structure features are modelled with elements. Behind each node a set of actions is dealing with tasks such as: (1) retrieval of structural information of a body, (2) putting of intermediate modelling results to FEMM and (3) modification of preceding results if needed.

8.5.2 Reasoning

Reasoning is guided under the reasoning strategy to match individual rules. Rules in the GMMFEM system represent association-based knowledge, e.g. IF (body_type='bar') THEN (direct the reasoning process to the bar-body reasoning block), and IF (bar-body has all its structure features \( c_j(j=2,3,4,5)=0 \)) AND (the purpose of modelling is for kinematic analysis) THEN (assign a binary element to the whole bar body). The first rule is a kind of meta rule in the sense that it directs the further use of other rules, while the second rule infers a
conclusion. In general all rules in GMMFEM system are represented at the conceptual level in the following form.

- Rule identifier:
- Purpose:
- Condition:
- Conclusion:
- Action:

**Example:**

Rule identifier: R0001.

Purpose: Assign element for a bar body-i.

Condition:
1. Task = 'kinematic analysis ∧
2. $c_i(j = 2, 3, 4, 5) = 0$.

Conclusion:
1. There must be a Binary element chosen for bar-i.

Action:
1. Set the status variable as number '2'.
2. Record the information that bar-i is modelled with a binary element

At implementation level rules may be written in a declarative manner, i.e. rules are coded as data. In the GMMFEM system, meta rules are usually represented in a procedural manner, while non-meta rules may take a declarative manner. For example, a set of rules reasoning upon the structure feature of a bar body is represented in an ABC-table defined below (ABC
is a main memory DBMS and was introduced in Section 3.5 of Chapter 3):

**TABLE rule_for_bar**

<table>
<thead>
<tr>
<th>ATTRIBUTE</th>
<th>rule_id:</th>
<th>INTEGER (KEY)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTRIBUTE</td>
<td>package_name:</td>
<td>CHARACTER</td>
</tr>
<tr>
<td>ATTRIBUTE</td>
<td>task_type:</td>
<td>CHARACTER</td>
</tr>
<tr>
<td>ATTRIBUTE</td>
<td>feature_item_2:</td>
<td>INTEGER</td>
</tr>
<tr>
<td>ATTRIBUTE</td>
<td>feature_item_3:</td>
<td>INTEGER</td>
</tr>
<tr>
<td>ATTRIBUTE</td>
<td>feature_item_4:</td>
<td>INTEGER</td>
</tr>
<tr>
<td>ATTRIBUTE</td>
<td>feature_item_5:</td>
<td>INTEGER</td>
</tr>
<tr>
<td>ATTRIBUTE</td>
<td>status:</td>
<td>INTEGER</td>
</tr>
<tr>
<td>ATTRIBUTE</td>
<td>element_type:</td>
<td>CHARACTER</td>
</tr>
<tr>
<td>ATTRIBUTE</td>
<td>action_name:</td>
<td>CHARACTER</td>
</tr>
</tbody>
</table>

**END_TABLE**

**Instances:**

<0001, 'RUNMEC', 'Kinematics', 0,0,0,0,0,'Binary','Create_one_binary_on_body'>
<0002, 'RUNMEC', 'Kinematics', 1,0,0,0,0,'Binary', 'Create_more_binary_on_body'>

A general procedure manages this table in such a way that structure features of a particular body-i are matched with the table and appropriate action procedures are inferred. The matching procedure is repeatedly carried out. During the matching procedure, status and structure feature values are updated. In the following an algorithm is presented (the syntax of the algorithm follows what was introduced in Chapter 6):

**Assume:**

- 'lid' denotes the instance identifier.

**Algorithm:**

**REPEAT**

Set 0 to 'lid'

Search table to match the instance;

Put instance identifier to 'lid';

IF ('lid' > 0) THEN

Infer the conclusion and execute the action routine

Set the respective $c_{ij} = 0$ {This is because the respective feature has been modelled.}

IF ($c_{ij}=0$, $j=2,3,4,5$) THEN

$s_i=2$
ELSE
  s_i=1
END_IF
END_IF
UNTIL (no instance is matched)

An example is shown in Fig.8.22 to demonstrate how the algorithm works. The idea of such rule-tables with its algorithm can make the program more flexible and general in the sense that change of any rule won't require change of source code.

The kind of rules illustrated above corresponds to knowledge base (A). So those rules are used to instantiate the generic part of FEMM. The knowledge base (B) is about specific conventions which are only applicable to individual FEM-based systems such as RUNMEC. This kind of knowledge is implicitly represented in the program source code in the current version of GMMFEM, albeit some parts of them can be represented in a declarative manner to achieve more flexibility of the system.

![Diagram showing evolution of structure features vs reasoning process](image)

*Figure 8.22. Evolution of Structure Features vs Reasoning Process*

8.6 The System Demonstration through Cases

The testing of the GMMFEM prototype system is carried out with several examples. The task of all these examples is kinematic analysis or simulation. They are tested in the KINANA environment in the sense that they are first edited with the EDMECH system (Chapter 7), then the GMMFEM system is triggered to generate a corresponding input data file of RUNMEC, and finally RUNMEC is triggered to produce a file with all moving coordinate values \( x^i \) (zero-order, 1st-order and 2nd-order quantities). The GMMFEM system also makes the FEMM locally persistent so that if necessary the FEMGMM system can take GMM, FEMM and calculated \( x^i \), as resources, to produce the user-oriented output. Finite element models of RUNMEC for all these examples are generated in less than 3 seconds in a 486 (66-MHz) personal computer. Discussions of solving these examples without GMMFEM are also given.
Example-1. A composite gear-bar mechanism (Fig.8.23a)

The mechanism is used in a textile machine in transferring a kind of material (Hua & Tan, 1985). The requirement of transfer is the uniform step motion of the output. The feature of the output motion to have a step motion is: $\beta(0) \neq \beta(2\pi)$. For the purpose of the motion simulation the mechanism is sketched with EDMECH as discussed in Chapter 7, Fig.7.15. The finite element model of RUNMEC is generated with the GMMFEM system and is graphically shown in Fig.8.23b. Table-8.2 gives an impression of the input data file for RUNMEC. While a detailed description of the model of RUNMEC is out of purpose here, a brief interpretation is given in table-8.3.

Table-8.2. The finite element model of RUNMEC for example-1

<table>
<thead>
<tr>
<th>KIN ANA</th>
<th>; kinematic analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOPOLOGY</td>
<td></td>
</tr>
<tr>
<td>ELEM</td>
<td>BIN      4 14 15 14 15</td>
</tr>
<tr>
<td></td>
<td>HBE      14 15 8</td>
</tr>
<tr>
<td></td>
<td>BIN      14 15 18 19</td>
</tr>
<tr>
<td></td>
<td>BIN      18 19 16 17</td>
</tr>
<tr>
<td></td>
<td>BIN      16 17 6 7</td>
</tr>
<tr>
<td></td>
<td>HBE      16 17 9</td>
</tr>
<tr>
<td></td>
<td>GR1      10 9 11 1</td>
</tr>
<tr>
<td></td>
<td>HBE      18 19 12</td>
</tr>
<tr>
<td></td>
<td>GR1      13 12 10 2</td>
</tr>
<tr>
<td></td>
<td>GR1      8 12 13 3</td>
</tr>
<tr>
<td>FORM</td>
<td>ANGB     1 1 1.0</td>
</tr>
<tr>
<td></td>
<td>ANGB     2 1 -1.0</td>
</tr>
<tr>
<td></td>
<td>ANGB     4 2 1.0</td>
</tr>
<tr>
<td></td>
<td>ANGB     3 2 -1.0</td>
</tr>
<tr>
<td></td>
<td>ANGB     5 3 1.0</td>
</tr>
<tr>
<td></td>
<td>ANGB     6 3 -1.0</td>
</tr>
<tr>
<td></td>
<td>ANGB     4 4 1.0</td>
</tr>
<tr>
<td></td>
<td>ANGB     8 4 -1.0</td>
</tr>
<tr>
<td></td>
<td>LENL     1 5 1.0</td>
</tr>
<tr>
<td></td>
<td>LENL     3 6 1.0</td>
</tr>
<tr>
<td></td>
<td>LENL     4 7 1.0</td>
</tr>
<tr>
<td></td>
<td>LENL     5 8 1.0</td>
</tr>
<tr>
<td></td>
<td>PARG     7 9 1.0</td>
</tr>
<tr>
<td></td>
<td>PARG     9 10 1.0</td>
</tr>
<tr>
<td></td>
<td>PARG     10 11 1.0</td>
</tr>
<tr>
<td></td>
<td>ANGB     1 12 1.0</td>
</tr>
</tbody>
</table>
Section 8.6

The System Demonstration through Cases

GEOMETRY
XFIX 1 -.36, 2 -.70, 3 -.50, 4 .00, 5 .00, 6 8.50, 7 .00
XMOV 8 .00, 9 -1.6952, 10 .00, 11 .00, 12 .00, 13 .00, 14 3.50
      15 5.00, 16 9.50, 17 8.00, 18 6.0, 19 5.0
PARA 1 .00, 2 .00, 3 .00, 4 .00, 5 6.10, 6 3.00, 7 3.70, 8 8.30,
      9 .00, 10 .00, 11 .00, 12 .00

STORAGE
BUFE 12 101
BUFX0 11 102
BUFKX1 11 103

BEGINPIC
PLOTFUNC 101 102
PLOTFUNC 101 103
ENDPIC

ANIMATE 2.6 film
MOVEMENT
MOV_RELE 72 0.087

END

Figure 8.23. Example-1: a Bar-Gear Mechanism
Table 8.3: The illustration of the finite element model of RUNMEC for example-1

<table>
<thead>
<tr>
<th>element type</th>
<th>relevant body</th>
<th>where</th>
</tr>
</thead>
<tbody>
<tr>
<td>binary (P6)</td>
<td>8</td>
<td>A-&gt;B</td>
</tr>
<tr>
<td>half beam (P10)</td>
<td>1</td>
<td>B-&gt;</td>
</tr>
<tr>
<td>binary (P6)</td>
<td>6</td>
<td>B-&gt;E</td>
</tr>
<tr>
<td>binary (P6)</td>
<td>6</td>
<td>E-&gt;D</td>
</tr>
<tr>
<td>binary (P6)</td>
<td>7</td>
<td>D-&gt;C</td>
</tr>
<tr>
<td>half beam (P10)</td>
<td>7</td>
<td>D-&gt;</td>
</tr>
<tr>
<td>gear(I) (P9)</td>
<td>3,5</td>
<td>D-&gt;C</td>
</tr>
<tr>
<td>half beam (P10)</td>
<td>6</td>
<td>E-&gt;</td>
</tr>
<tr>
<td>gear(I) (P9)</td>
<td>4,3</td>
<td>E-&gt;D</td>
</tr>
<tr>
<td>gear(I) (P9)</td>
<td>1,2</td>
<td>B-&gt;E</td>
</tr>
</tbody>
</table>

1: the motion of gear-5
2: the motion of bar-7

Figure 8.24: The Output Motion of Gear-5 and Bar-7

Fig. 8.24 exhibits several outputs, where the motion of gear-5 is indeed non-periodic while the other motions are periodic. Some snapshots of the motion animation are displayed in Fig. 8.25. It has been shown that the initial configuration of the mechanism can be given in quite a rough sketch (Fig. 8.25a); in the geometrical sense three nodes (B,D,E) are even not on one line. (The gear body is not shown in Fig. 8.25.) The adjustment of this initial position
to the initial position which meets the kinematic parameters is done correctly (Fig. 8.25b).

![Diagram](image)

**Figure 8.25. The Motion Animation of the Mechanism in Example-1**

An experiment was conducted at the China Textile University (1988) that a student after studying the course: *Mechanism Analysis and Design* (80 hours) would take on average three days to derive the formulae to calculate the motion of gear-5. With the derived formulae, a team of students (about 20) needs to spend half a day to work together; they each calculate β5 for one position of the crank angle, with a non-programable calculator. The complete motion curve (Fig. 8.24) can then be drawn.

Using the finite element approach at the Delft University of Technology of The Netherlands (nowadays), a student after studying both the finite element and mechanisms courses would invest once 20 hours to study the manual of RUNMEC (perhaps in some discussions with his/her coach) to make the model in a few hours. Using the system now, this modelling can be done in a few minutes with the mechanism editor. The FEM-model can be automatically generated by GMMFEM.
Example-2. A composite cam-bar mechanism (Fig.8.26a).

Such a cam-link mechanism is very useful in practice (Hua & Tan, 1985; Eventoff, 1992). The mechanism can be entered into the computer with EDMECH. In this mechanism the input motion of the mechanism, the cam motion, has non-equidistant steps. These steps are read from a data file, here INPTFILE.DAT. The cam profile is also given in a file. The input data file of RUNMEC is generated as shown in table-8.4. Fig.8.26b also displays one mechanism position other than the initial position.

Table-8.4. The finite element model of RUNMEC for example-2

KIN ANA ; kinematic analysis
TOPOLOGY

<table>
<thead>
<tr>
<th>ELEM</th>
<th>BIN</th>
<th>1</th>
<th>2</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>2</td>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>12</td>
<td></td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td>6</td>
<td></td>
<td>8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FORM</th>
<th>LENL</th>
<th>1</th>
<th>1</th>
<th>1.0</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>2</td>
<td></td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>3</td>
<td></td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>4</td>
<td></td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>3</td>
<td></td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>4</td>
<td></td>
<td>-1.0</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>5</td>
<td></td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>5</td>
<td>7</td>
<td>1.0</td>
<td></td>
</tr>
</tbody>
</table>

GEOMETRY

<table>
<thead>
<tr>
<th>XFIX</th>
<th>0.0, 2 0.0, 3 6.0, 4 0.0, 5 9.0, 6 0.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>XMOV</td>
<td>7 0.0, 8 9.0, 9 -1.0, 10 6.0, 11 4.0, 12 0.0, 13 4.0</td>
</tr>
</tbody>
</table>

CAM 5 CAM1.DAT ; CAM1.DAT is for the profile of a cam

STORAGE

ANIMATE 1.6 STEP

MOVEMENT

MOV_FUNC INPTFILE.DAT ; INPTFILE.DAT is for non-equal distance step

END
8.7 Discussion and Conclusions

Automating finite element modelling in the context of GMM and EDMECH is a necessary means to achieve more intelligence of CAD/CAM of mechanical systems. Since the finite element approach to the kinematic and dynamic analysis and synthesis of mechanisms is the most general one, a CAD/CAM system which takes this approach as the basis will be more promising. The generality of finite element approach lies in its way of formulation of the constrained motion equation with the continuity equation. Other approaches more or less have some limitations in their own right. For example, Chieng & Hoelzel (1988) admitted that their method was not applicable to any type of mechanism and restricted to only bar linkages. The
approaches based on the closed loop matrix (Yang and Freundenstein, 1964; Sheth & Uicker, 1972) would have to find special techniques to handle a mechanism which contains a gear or a cam, because the constraints between bodies introduced by these machine bodies are usually not represented in the coordinate transformation matrix.

The system to perform the finite element modelling automation is complex and must take the so called model-based approach. This approach could be viewed as a combination of IF-THEN reasoning and data management. The data management is used for the models of underlying domain object structures and principles of behaviours of the domain object (e.g. the finite element principle). The knowledge represented with the IF-THEN formalism is used to 'diagnose' the domain object structure. Conditions and conclusions are viewed as input and output of an IF-THEN rule, respectively, and they are both related to the model regarding the domain object (structures and principles of behaviours). Fig.8.27 depicts with no further remarks an analogy of two things: (i) the relationship between the model of the domain object and the mathematic equation to model the behaviour of the domain object and (ii) the relationship between the model of the domain object and the IF-THEN formalism to reason upon the domain object. Such an observation implies an inevitable need of the IF-THEN formalism in a model-based system. This implication is identical with that by Keller (1987) who commented: IF-THEN is a major tool in representing knowledge in eventually every expert system which has been done, or now being done, or is likely to be done for some time. In the viewpoint of this work it might be more robust to skip off the word 'major' from this statement.

![Diagram](image)

*Figure 8.27. Analogy: Behaviour Function vs IF-THEN formalism for Reasoning*
Design Synthesis Tools in CIMOME: Implementation Overview and Perspective

9.1 Introduction

Applications of the theory described in Chapter 2 and Chapter 4, respectively, will be shown in this chapter with the discussion of several implementations of CIMOME which have been or are being carried out. These implementations concern sub-environments of CIMOME, in which several design synthesis tools are brought together. Detailed descriptions are not aimed; only theoretical points elaborated in some previous chapters will be highlighted. These include the environment architecture design principle (Chapter 2), the general framework of the product model and the design process management (Chapter 4). Another purpose of this chapter is to predicate the perspective of CIMOME by exhibiting some functionalities achieved from the current implementation.

Section 9.2 deals with a sub-environment called CAMDES. CAMDES binds the tools TADSOCC (see Chapter 2 for definition), GMMFEM (Chapter 8), FEMGMM (Chapter 8), RUNMEC (Chapter 8) and EDFUNC (see Chapter 2 for definition). Design automation and user-friendliness in the cam mechanism design (both single cam and compound cam mechanisms) are considerably improved. The discussion also makes more explicit the CIMOME architecture described in Chapter 2. The implementation of CAMDES is a model for the remainder of design synthesis tools. Next two sections will briefly introduce the implementation of two other design tools: one is DSYNOP (Section 9.3) and the other is LINKDES (Section 9.4). Section 9.5 concludes this chapter.
9.2 CAMDES: a Sub-Environment for the Compound Cam Mechanism Design

The single cam mechanism is composed of a cam, a bar and a carrier which supports the cam and the bar, see Fig.9.1, where several types of cam mechanisms are shown. In general the cam, as an active body, performs input while the bar, as a follower, performs output. The main issue of the cam mechanism design is to calculate the cam profile to meet the behaviour function between the cam rotation and the follower displacement. The calculation of the cam profile needs first to prescribe a set of kinematic parameters such as B,D and R0 as shown in Fig.9.1. Sets of these kinematic parameters depend on types of the cam mechanisms. For example the parameters for the type of cam mechanisms shown in Fig.9.1b are: E and R0. In design practice these kinematic parameters may be given by the designer, in accordance with some constraints such as the working space of the mechanism in a machine, or can be calculated for some constraints such as allowable pressure angles (at rising stroke and at returning stroke).

![Diagram of single cam mechanisms](image)

**Legend:**  
- fixed node;  
- prime circle of cam;  
- roller

**Figure 9.1. Single Cam Mechanisms**

The compound cam mechanism adds extra bodies and joints on the single cam mechanism and the concerned behaviour function is naturally not those between the cam and its follower, see Fig.9.2. The extra part in a compound cam mechanism is assumed to be known and can include any type of bodies and joints. In addition to the kinematic parameters such as B,D,R0, interface measure (like angle λ in Fig.9.2) on the follower must be concerned.

There are basically two approaches to the cam design process of the compound cam mechanism. The first is based on the kinematic inversion. Assuming there is a reversed rotation (quantitatively identical to the cam rotation) acting on the whole mechanism, the cam stands still (the 'frame'). The whole mechanism has two input motions (one is through the original frame body and the other the output body). The trajectory of the point on the follower driven with these two input motions is the cam profile. To the compound cam mechanism, care should be taken:
The whole extra part of the mechanism is inverted instead of only follower in the case of the single cam mechanism.

The output motion performed with the follower in the case of the single cam is now replaced by the body (see Fig. 9.2, body-5) which actually performs the output motion.

![Diagram of a compound cam mechanism]

**Legend:** motion measurement reference line on the ground; $\alpha$: input; $\beta$: overall output; $\delta$: output of the cam follower; $\lambda$: interface angle

**Figure 9.2. A Compound Cam Mechanism**

The second approach requires decomposition of the whole mechanism into two parts: a single cam mechanism part and an extra part. The desired behaviour function, $\beta = \beta(\alpha)$, see Fig. 9.2 is converted to the behaviour function between the cam and its follower, i.e. $\delta = \delta(\alpha)$. After this treatment the remainder of the compound cam mechanism design becomes the matter of the single cam mechanism design.

The second approach is considered better, based on the following reasons.

- The kinematic parameters $(B, D, R0, \lambda)$ are separated from those of the extra part of the mechanism. This is significant because essentially the kinematic parameters $(B, D, R0, \lambda)$ are a kind of design variables but those of the extra part are known.
- The knowledge of the single cam mechanism design can be reused. For instance the process that can determine the minimal radius of the *prime circle* given the allowable pressure angle and the parameter D or B - a well known method in the cam mechanism design domain (Hua, 1983; Hua & Tan, 1985). Remarkable is that the general-purpose commercial package such as CAMLINK based on the first approach failed to reuse this well-known approach. For the purpose to control the maximal pressure angle CAMLINK uses an optimization scheme, which is unnecessary in our system CAMDES.

### 9.2.1 Extension of GMM to the cam mechanism

The data representation of the cam mechanism is a necessary step to have an intelligent computer support - an observation which can be drawn from the discussion of the previous
chapters (5,7,8). Naturally GMM, which has been established in Chapter 5, is considered for some extensions for this purpose. The first extension is to define a class type CAM as shown in Fig.9.3, where instances considered at the implementation level are also shown. It should be noted that the CAM class type is a sub-type of type MECH-BODY (Chapter 5), which has been introduced earlier in Section 5.5 of Chapter 5, particularly in Fig.5.9.

The second extension is to represent the specialized properties of the connection between the cam and the bar follower, which has partially been discussed in Section 5.5 of Chapter 5, particularly see Fig.5.18. The class type CAM-FOLLOWER is now formally defined as shown in Fig.9.4, where instances considered at the implementation level are also shown.

The extension of GMM to the data representation of the cam mechanism semantics once more demonstrates the extensibility of GMM. Clearly GMM with its extension to the cam enables to represent the compound cam mechanism in the computer. Fig.9.5 shows the situation where two mechanisms discussed earlier in Chapter 7, see particularly Fig.7.15 and Fig.7.16, are put together with EDMECH. The implications are, on one hand, that the extra part of a
compound mechanism is not limited to the bar mechanism (Fig.9.5a), and on the other hand, that a designed cam mechanism can be put in a machine context (Fig.9.5b) for further processing.

It must be noted that the mechanism shown in Fig.9.5 is explicit to both the designer (the end-user) and the computer. *Explicit to the designer* is meant for the kinematic mechanism diagram on the screen. *Explicit to the computer* is meant for GMM. The consequence of these two kinds of explicit representation is that further design activities intended by the human designer can be readily carried out. In the viewpoint of the cam mechanism design, it means that after the cam profile is obtained the designing of the cam mechanism can be continued. This includes modification of the mechanism structure by e.g. adding more machine bodies on the cam mechanism or adding mass properties for dynamic analysis.

9.2.2 GMM versus a cam design parameter model for the cam mechanism

A design parameter model is defined as a data representation of a set of parameters which serve as input and/or output to a design algorithm. For the cam mechanism design, the design parameter model includes the information about B,D,R0, etc. dependently of types of cam mechanisms. A design parameter model is needed based on the following reasons:
The design parameter model inherently facilitates the user-computer interaction at the level of individual design algorithms. For example, to the algorithm for the cam profile calculation, the set of parameters (B,D,R0, etc) is explicit. To improve the user-computer interaction it will be better to make them explicit to the end-user too. (This is the same philosophy initiating the development of GMM, see Section 5.2 of Chapter 5.) Fig.9.6 exhibits a window of CAMDES to give an impression of the user-computer interaction on this set of parameters owing to a cam design parameter model behind the system. Both the kinematic mechanism diagram (in the drawing area) and the parameters (on the right menu window) are displayed; so the parameters are explicit to the end-user. Furthermore to non-experienced users the relationship of the parameters with the kinematic mechanism diagram may not be so clear. Fig.9.6 shows that, while the end-user points the mouse cursor to the text entry box for parameter R0, the corresponding body in the drawing area is highlighted and help messages are appeared in the message area to remind the end-user of the consequence of his further action - change of the value of R0.

It is understood that there could be more than one design solution to a chosen type of cam mechanisms. Different design solutions need to be represented and stored in the computer. Two schemes, with or without a design parameter model, are, respectively, shown in Fig.9.7. Clearly without the design parameter model three instantiated GMMs are needed, which leads to complex data management and much overhead. (Three GMMs correspond to three instantiated databases while three sets of design parameters correspond to different instances of one design parameter model.)
The presence of a design parameter model will be more efficient because change of design parameters does not immediately affect GMM.

The designer may want to terminate his current design session but to keep the current design state while a final solution is not yet reached. This is only possible when design parameters can be easily made persistent. The database model of design parameters is therefore very helpful.

The conceptual schema of a cam parameter model is shown in Fig.9.8. The model schema
is generally organized in two levels: a generic class type CAM-DESIGN-PARAMETER and its sub-class types. In the generic class, common parameters of any type of cam mechanism are defined while in sub-classes specialized parameters of a specific type of cam mechanism are defined. The model defined as such enables easy extension. In the model schema the relationship between the design parameter model and GMM is represented in the attribute MECHANISM. At the implementation level, this attribute stores an individual mechanism identifier. A design solution corresponds to a set of design parameters which are further stored as instances of the model schema. (In this way the scheme as described in Fig.9.7a is realized.)

![Diagram of CAM-DESIGN-PARAMETER and RR-roller]

Legend:  
[α]: allowable pressure angle during rising stage;  
[α]': allowable pressure angle during descending stage;  
RR-roller: cam: rotation, follower: rotation, contact form: roller

**Figure 9.8. Data Representation of Cam Design Parameters**

\[ DL (GMM \rightarrow DPM) \]

GMM \[ \rightarrow \]

\[ DL (DPM \rightarrow GMM) \]

**Figure 9.9. Data Link between GMM and Cam Design Parameter Model**

As mentioned earlier, GMM and the design parameter model of the cam mechanism are related. The design parameter model is a kind of view of GMM according to the general framework of product model proposed in Chapter 4. The information contained in this view is meaningful only within the context of the object - GMM in this case. The design of a compound cam mechanism, i.e. modification of the instances of GMM is, however, through this view. (Here a kind of case study is actually given concerning the object and object view concept in the framework of product model proposed in Chapter 4.) Two program processes (see Fig.9.9) are needed between GMM and the design parameter model to realize the information exchange between the two models. These program processes belong to the category of the *data link with attached process* - a concept identified earlier in Section 2.3 of
Chapter 2 (in the next section more discussions are given).

9.2.3 Data communication scheme in CAMDES

*Define:*

- D1: Data block concerning a complete compound cam mechanism represented with the GMM-format (Chapter 5);
- D2: Data block concerning a complete behaviour function represented with the GFM-format (Chapter 4);
- D3: Data block concerning a behaviour function represented in a discrete point manner with the format of a simple programming file below:

\[
\begin{array}{cccc}
  x & y & y' & y'' \\
  - & - & - & - \\
  - & - & - & - \\
  - & - & - & - \\
\end{array}
\]

- D4: Data block concerning the design parameters represented with the data format of the cam design parameter model (see Section 9.2.2);
- D5: Data block concerning the single cam mechanism represented with the GMM-format;
- D6: Data block concerning the mechanism part excluding the single cam mechanism represented with the GMM-format;
- D7: Data block concerning the cam design parameter in a data format known to TADSO (Van den Berge, 1992);
- ST-1: A Service Tool (originally TADSO) performing the task to calculate the design parameters and/or the cam profile;
- ST-2: A Service Tool performing the task to convert the overall behaviour function of a compound cam to the behaviour function between the cam and its follower;
- DL-1: A Data Link process which generates the behaviour function (D3) from the behaviour function (D2);
- DL-2: A Data Link process which divides a compound cam mechanism into a single cam mechanism and an extra part;
- DL-3: A Data Link process which generates a design parameter model from GMM and updates GMM from a design parameter model, which corresponds to two data link processes mentioned in Fig.9.9.
- DL-4: A Data Link process which converts the cam design parameters from data block (D4) to data block (D7).

The data communication scheme of CAMDES is shown in Fig.9.10. Further remarks are given below:
In CAMDES both GMM and the design parameter model are needed in the design process for the design parameter calculation and the cam profile calculation, so data link process (DL-3) is not made physically external (see the discussion of Chapter 2, particularly Section 2.4.2). (GMM is needed in the above mentioned design process to graphically interact with the end-user regarding the kinematic mechanism diagram, e.g. as implied in Fig.9.6.)

Service tool ST-1 (originally TADSOOC) needs the behaviour function as a precondition. The decision that it does not directly communicate with data block (D2) but with (D3) is in accordance with the data integration guidelines (I)-(IV) earlier proposed in Chapter 2. Data link process (DL-1) is made physically external because EDFUNC does not operate on data block (D3). Service tool ST-1 (TADSOOC) works on data blocks (D3) and (D7). By doing so the original tool TADSOOC needs only minor adaption. In the implementation practice the two systems EDFUNC and CAMDES were actually in parallel developed. This fact readily proves the usability of the data integration guidelines to develop a better environment architecture development, as was elaborated in Section 2.3 of Chapter 2.

- Data link process DL-2 is made physically external.
- Service tool ST-2 includes several processes (tools). The main steps of ST-2 are outlined:

  - Generate a finite element model for the part excluding the cam and follower via GMMFEM (see Chapter 8).
  - Calculate the finite-element coordinates of this part of mechanism via RUNMEC (see Chapter 8).
  - Generate the function \( \delta = \delta(\beta) \) given \( \beta_i = \beta(\alpha_i), i = 1, 2, \ldots, n \), where \( n \) denotes the number of positions of the cam, see Fig.9.2. This is done via FEMGMM (see Chapter 8)
  - Formulate the function \( \delta = \delta(\alpha) \) with the data format of data block (D3).

The input/output data related to service tool ST-2 are local persistent and their data formats were described in the previous parts of this dissertation, e.g. Chapter 8, etc.

The data communication scheme in CAMDES can be thought of as a case study of the theoretical description of the CIMOME architecture in Chapter 2. The concept of design
Section 9.2 CAMDES: a Sub-Environment for the Compound Cam Mechanism Design

parameter model and its relationship with GMM is found to be applicable to all design synthesis tools in CIMOME.

![Diagram](image)

*Figure 9.11. Data Representation of Project*

9.2.4 The project database vs CAMDES
CAMDES also considers the possible case that there is no satisfactory solution found. The design process should then be directed to one of the following states:

- modification of the extra part of the mechanism with EDMECH,
- modification of the overall behaviour function with EDFUNC, and
- returning to the CIMOME top level to consider other mechanisms, with or without a cam.

The important question to support the above is how to keep the history information about the design. For example, it might be possible that the designer has specified more than one behaviour function and now he wants to modify the one which is not currently active. This situation will equally happen to the mechanism. It might also be possible that after the designer tries out the solution without the cam he finds that the cam solution, though not satisfactory, is better than others and he then wants to return to the cam design session, i.e. the CAMDES environment. The discussion poses a need to have some coordinations among design synthesis tools and EDFUNC and EDMECH as well. Such coordinations further require to have a so-called *project database* defined at the CIMOME top level, which corresponds to the environment database as earlier mentioned in Section 2.4 of Chapter 2. The project database schema is defined in Fig.9.11. It is noted that the project database
actually contains some meta information about the mechanism and the behaviour function. *e.g.* the database file name and the version number. The reasons to include the information about the database file name are:

- In the ABC system, as earlier mentioned in Section 3.5 of Chapter 3, objects of one database are clustered into one file. That is to say, for instance, a complete mechanism, an instantiated GMM, is stored in one file in the secondary memory; so one mechanism corresponds to one file with GMM-format.
- In run-time there are only one mechanism and one behaviour function in the main memory. The file name is therefore a means to keep connection with all the mechanisms and the behaviour functions concerned in one project.

Fig. 9.12 shows two windows of CAMDES which allows the scenario that the designer wants to modify the part of the mechanism excluding the cam. For this purpose the designer uses the mouse to point to menu button 'Edit' and button 'Mechanism' in Window-1 of CAMDES sub-environment. After that the design process is directed to Window-2 of EDMECH, where a bar body is selected for updating. If there is more than one mechanism in the current project the designer will be confronted with the choice of all these mechanisms in addition to the current active mechanism. Naturally after returning to CAMDES environment the current active mechanism may be changed. (Returning from Window-2 of EDMECH to Window-1 of CAMDES is done via pointing to button 'File' and then button 'Exit'.)

From the discussion above, it can be concluded that the project database is maintained at the CIMOME level and the design tool environment levels as well, see Fig. 9.13. That the project database is maintained at the CIMOME level makes sense when the design process is switched from one tool (say Tool-A) environment to another tool (say Tool-B) environment. Because the communication between design tools (except for EDMECH and EDFUNC) must be enacted via the CIMOME top manager, as discussed in Section 2.4.2 of Chapter 2, it is the responsibility of the CIMOME top manager to provide necessary information to Tool-B through the process/process protocol described in Section 2.4.2 of Chapter 2. Suppose Tool-A is CAMDES and Tool-B is LINKDES. It makes no sense to provide the information of the cam mechanism and the cam design parameters to the LINKDES environment; it is necessary, however, to provide the required behaviour function to the LINKDES environment.

**9.2.5 Computer-aided design process management in CAMDES**

In Chapter 2 the design process management in an integrated environment was elaborated. In Chapter 4 a design process model was described. The experiment of the design process management in CIMOME has been carried out in the implementation of CAMDES. In the following, this experiment is described.

The process model for the experiment has made the following simplifications with respect to
Section 9.2 CAMDES: a Sub-Environment for the Compound Cam Mechanism Design

the model proposed earlier in Section 4.5 of Chapter 4:

**Figure 9.12. Switch between EDMECH and CAMDES**

- Each task/action corresponds to one computer module (a program or a procedure).
- Predecessor tasks/actions and required resources necessary for a task/action to be initiated
(see Section 4.5 of Chapter 4) are put together as *precondition* of this task/action.
- Successor tasks/actions and provided resources necessary after execution of a task/action (see Section 4.5 of Chapter 4) are put together as *postcondition* of this task/action.
- Determination of the required resources considers two cases:
  - Whether or not a database file of a mechanism exists and the input/output motion quantity is defined. (This shows that determination of required resources is quite application domain specific. A general implementation may be impossible.)
  - Whether or not a database file of a behaviour function exists.

![Figure 9.13. The Project Database vs Design Synthesis Tools and CIMOME](image)

**Figure 9.13. The Project Database vs Design Synthesis Tools and CIMOME**

![Figure 9.14. A Process Model for Compound Cam Mechanism Design](image)

**Figure 9.14. A Process Model for Compound Cam Mechanism Design**

Based on these simplifications three tables are defined, see Fig.9.14. The remarkable ones are the *precondition table* and the *postcondition table*. The attributes of these two tables reflect the condition aspects that are currently concerned in CAMDES. The value of the attributes of all condition aspects is 0 or 1 or -1. Their meanings:
Section 9.2 CAMDES: a Sub-Environment for the Compound Cam Mechanism Design

- 1: the condition aspect must be met,
- 0: the condition aspect must not be met,
- -1: the condition aspect is not significant

Possible instances of these table are shown below to have an impression.

In Task/action table:

<T_001#, PreC_1#, PostC_1#, 'Ready'>,
<T_002#, PreC_2#, PostC_2#, 'Not-ready'>, where first attribute value is object id.

In PreCondition table:

<PreC_1#, -1, -1, -1, -1, -1>
<PreC_2#, -1, -1, -1, -1, -1>, where first attribute value is object id.

In PostCondition table:

<PostC_1#, 1, -1, -1, -1, -1>, where first attribute value is object id.

A design state vector (SV for short) is defined below:

$$SV::=(\text{condition aspect-1, condition aspect-2, \ldots, condition aspect-n}), \text{ here } n=5.$$  

Suppose T_001# denotes the task to define a cam mechanism type and initially SV gets the value as \(<-1, -1, -1, -1, -1>\). The above instances infer the following meaning:

- Before any action is done a mechanism type must be defined for the cam mechanism design.
- After a mechanism type is defined SV should be updated as \(<1, -1, -1, -1, -1>\).

Fig.9.15 shows two windows of CAMDES. One is conventionally used and hereafter called design window. The other is a special one called process window, proposed in CAMDES, where a prescribed design process and its proceedings are explicitly displayed. In process window, button 'Select' is used to select one icon of tasks/actions. Button 'Help' can then be pressed to get help messages of the current task, e.g. how to initiate this task in design window. Two algorithms generally maintain the design process. One is to infer all the next-step tasks/actions and the other is to handle the user-action. These two algorithms are briefly described.
Algorithm-1:

Let:

'Pre_Ins' denotes the instance id of 'Precondition' table.
'Task_Ins' denotes the instance id of 'Task/action' table.

Algorithm:

Set 'Pre_Ins' to 0;
Search 'Precondition' table with 'SV';
Put matched instances into 'Pre_Ins';
IF ('Pre_Ins' is not Empty) THEN
    Search 'Task/action' table with 'Pre_Ins';
    Put matched instances into 'Task_Ins';
    Update the value of attribute 'Current-status' of 'Task/action' table to 'Ready';
    Display the 'Ready' signal on the process window for these tasks/actions.
END_IF

Algorithm-2:

Algorithm:

Get the user's action;
WHILE (action does not drop out the design process) DO
    IF (the action is one of 'Ready' tasks) THEN
        Update the value of attribute 'Current-status' of 'Task/action' table to 'In-active';
        Display this signal on the process window;
        Update 'SV' in terms of the instance of 'PostCondition' table corresponding to the current active task;
        Trigger the corresponding action routine to perform the action
    END_IF
    IF (the current action is finished) THEN
        Update the value of attribute 'Current-status' of 'Task/action' table to 'Done';
        Update the process window;
        Trigger algorithm-1 to infer the next task/action
    END_IF
END_WHILE

In Fig.9.15, the process window shows that now two tasks are ready to be initiated. One is to define mechanism and the other to define a behaviour function. Suppose the designer wants
Figure 9.15. Design Process Window and Design Window - 1

to initiate task-1. With possible help messages, he understands that he should proceed in the design window as shown, i.e. button 'File' -> button 'Retrieve' button 'Mechanism'. Fig. 9.16 shows that the selected task-1 has been done. The consequence is that in the design window a defined compound cam mechanism is displayed. In the process window the task execution chain is updated; next design task is to define a behaviour function. If the designer incidently ignores the explicitly displayed process chain, e.g. by initiating the task of cam profile calculation in the design window, the help message will appear, see Fig.9.17. In this way a design process will be made explicit to the designer.

The task/action which has been done can be invocable again, which implies change of the design. In this situation the designer will be confronted with the question such as whether the changed design should be kept in the system as an alternative. In this way an actual design process can be steered in an iterative manner, which is much close to the design process in practice. For the computer to have such a support, algorithm-2 in the above must be extended.
Figure 9.16. Design Window and Design Process Window - 2

There won't be fear that designers' role in such an environment is only passive. This conclusion is based on the following observations:

- Each task/action is initiated by the designer.
- There are some freedoms for the designer to make design decision. For example, the designer should decide either to specify the design parameters such as (D,B,R0) or to control them by the maximal pressure angle; see the two alternative design process routes in process window (Fig.9.16):

  (1) 1 -> 2 -> 3 -> 4 -> 6 or
  (2) 1 -> 2 -> 3 -> 5 -> 4 -> 6.

Another example is that the designer should first define a type of cam mechanisms without the control of the system, and the system can then calculate the profile of a cam.

With the increased knowledge about design and more sophisticated means to represent that
knowledge, freedom of the designer on some conventional tasks will be reduced. In the meantime, with the increased demand on products of high quality with lower developing cost and short lead time, the designer will be continuously given freedom to make design decisions which are fuzzy and yet difficult to be represented.

9.2.6 A design case demonstration

The reminder of the compound cam design process will be briefly overviewed with the example first shown in Fig.7.16 of Chapter 2 and discussed quite a lot in the above of the current chapter. The required behaviour function is defined (design task-2) with EDFUNC discussed in Chapter 4 and a window of EDFUNC is further shown in Fig.9.18. In Fig.9.19 the conversion of the overall behaviour function to the one between the cam and follower is shown, which corresponds to design task-3. The system will automatically detect the cam and the follower for the designer's confirmation, see Window-1 and Window-2, respectively. When there is more than one cam in the designed mechanism such confirmation is important because the system detects all cams. Design task-4 is then performed, see earlier Fig.9.6, supposing that the designer does not think the maximal pressure angle will be critical to his
Figure 9.18. Required Behaviour Function Defined with EDFUNC

mechanical system. In Fig.9.19, Window-3 shows that the designer initiates the cam profile calculation. The cam profile is calculated and displayed in Window-4. NC code of the cam profile is created via button 'Generate' -> button 'NC-code', see Window-4 of Fig.9.19. The connection to NC machines has already been one of the functionalities of original TADSOUC (Akker, 1988). Just to give an impression, in Fig.9.20 a special NC milling machine, sited at Lab of Mechanization of Production, with its code format, is shown.
Figure 9.19. Several Design Steps for a Case Demonstration
9.3 DSYNOP: a Sub-Environment for Dimension Synthesis via Optimization

As introduced earlier in Section 2.4.1 of Chapter 2, DSYNOP is mainly aimed at integrating a design synthesis tool called OPLAM in CIMOME. The capability of OPLAM is to perform the dimension synthesis via the optimization technique. As known, for such systems to be general to design problems, the key point is to facilitate the formation of an objective function and constraints on design variables, which are certain kinematic parameters in case of the mechanism design.

Because OPLAM uses Delft-FEM for the formulation of the objective function it becomes the most general one. However, its usability is restricted because the designer has to know Delft-FEM and he has to be involved in the detailed level of the specification of the input data file.
of OPLAM. This observation was also discussed in Chapter 8.

The general ideas behind this implementation are not difficult to be predicated:

- Make Delft-FEM invisible to the designer.
- Make it invisible to the designer that a dimension problem is formulated as an optimization problem. This means that the system will be generally divided into two levels: the user-level and the system level. At the user-level design problems are described invisible to the formulation of an optimization problem. At this level the feature-based requirement specification (see Section 4.4 of Chapter 4) is employed.
- The system level is further divided into two levels: system level-1 and system level-2. At system level-1, a dimension synthesis problem is converted to an optimization problem. At system level-2 an optimization problem is converted to the model which incorporates Delft-FEM with optimization technique.

9.3.1 GMM vs a design parameter model of DSYNOP
One of the key points to implement DSYNOP is to develop a design parameter model, which is philosophically similar to the situation of CAMDES. This design parameter model describes the kinematic parameters which are considered as design variables. For the dimension synthesis purpose, the information includes:

- the initial value of those kinematic parameters,
- the region within which the value of a kinematic parameter can be varied, and
- the new value of those kinematic parameters after the optimization - variation of those kinematic parameters to meet the objective function best.

The purpose to have a parameter model in DSYNOP is completely similar to that of the parameter model in CAMDES, e.g. to make the design parameters explicit both to the designer and the design algorithm. Fig.9.21 shows a window of DSYNOP, where in the drawing area a designed mechanism under design is drawn, while on the right menu window the design parameters are displayed. The same approach as CAMDES is used to make these design parameters more explicit to the designer. That is to say, while the designer points one parameter on the right menu window, the corresponding mechanism object will be highlighted simultaneously in the drawing area. Fig.9.21 also shows that extra design information concerning the design parameter, e.g. the varying region of a design parameter, will be stored in the design parameter model of DSYNOP.

Similar to CAMDES the design parameter model is a kind of object view of GMM. A data link exists between them. The data link process maintains the data consistency between GMM and the design parameter model. The data link process of DSYNOP differs from that of CAMDES. In CAMDES, the data link process works as a pure procedure. This is because
Figure 9.21. A Window of DSYNOP to Make Design Variable Explicit

Figure 9.22. Several Types of Kinematic Parameters Considered in DSYNOP

the feature of the design parameter of the cam mechanism is unique. For example, parameter (B) corresponds to the bar follower which is the unique bar that on one hand is connected with the cam via a cam joint and on the other hand is connected with the carrier via a revolute joint. However, DSYNOP is intended to design any type of bar and gear linkage rather than a cam mechanism. There are various types of kinematic parameters as shown in Fig.9.22, for example. It is then necessary to explicitly relate an individual parameter with a certain mechanism object in GMM. That is to say, the relationship between GMM and the design parameter model must be explicitly represented, see Fig.9.23.
Section 9.3  DSYNOP: a Sub-Environment for Dimension Synthesis via Optimization

Figure 9.23. Explicit Data Representation of Relationship between GMM and DSYNOP

Figure 9.24. Data Representation of Design Variable for DSYNOP

Fig. 9.24 shows a conceptual schema of a part of design parameter model of DSYNOP. The explicit data representation of the relationship between GMM and the design parameter model...
is found to be very helpful at the implementation level, e.g. to the simultaneous highlighting of mechanism objects while a design parameter is specified.

9.3.2 GFM vs the required behaviour model of DSYNOP
For the dimension synthesis via the optimization technique, some extra information on the required behaviour may be necessary. For example the tolerance between the required behaviour and the actual behaviour. Another example concerns the path generator, where the input motion position may be considered (or not). By following the philosophy used in the previous discussion of the relationship between GMM and the design parameter model of DSYNOP, a so-called Design Required Behaviour Model (DRBM for short) is particularly considered for the dimension synthesis via optimization. In GFM (Generic Function Model, see Chapter 4) only the common information to all design synthesis tools is represented.

In DRBM the extra information is structured and stored. A complete requirement model for the dimension synthesis via the optimization technique is a summation of GMM and DRBM. Fig.9.25 exhibits a window of DSYNOP where extra information on a required trajectory, i.e. tolerance and positions of the crank which performs the input motion, is specified.

9.3.3 Perspective of DSYNOP
At this moment the implementation of a prototype DSYNOP is carried out in particular with the path generator design problem (including the problem with derivative motion quantity). The implementation of the model at the user-level with a user-interface has been accomplished as implied in Fig.9.21 and Fig.9.25. The automatic formulation of an optimization problem for a path generator is straightforward. The automatic formulation of the Delft-FEM optimization problem largely depends on the automatic generation of a finite element model of mechanism for kinematics which has been demonstrated in Chapter 8.
9.4 LINKDES: a Sub-Environment for Type and Dimension Synthesis

As introduced earlier in Section 2.4.1 of Chapter 2, LINKDES is mainly aimed at integrating a design synthesis tool called TADSOL in CIMOME. TADSOL performs the task for type and dimension synthesis of link mechanisms. It works on a library of case mechanisms which would be solutions for a given behaviour function. The mechanisms, stored in the library, are classified in terms of the characteristics of the behaviour function that they will perform. In particular the characteristics are represented by a set of Fourier coefficients, obtained by harmonic analysis (Rankers et al., 1985). At the same time the required behaviour function can be analyzed and represented by a set of Fourier coefficients. TADSOL works as a matching process between these two groups of Fourier coefficients, so that mechanisms can be selected as potential solutions.

Two contexts are identified with respect to the mechanism, see Fig.9.26. One is the library context and the other is the machine context. Design requirements are described in the machine context because this is the only context known to the designer. An adjusting process
transfers the mechanism in the library context to the machine context, see Fig. 9.26. In the following several key steps in the implementation are briefly described.

![Diagram showing Library Context and Machine Context](image)

**Legend:** $\alpha$: input; $\beta$: output; $\rightarrow$ motion reference; $\uparrow$ global coordinate system.

*Figure 9.26. Two Contexts of the Mechanism: Library and Machine*

**Mechanism library database**
Kaper (1987) utilized a relational DBMS, ORACLE, to define a database of the mechanism library. This resulted in a bad performance. It would take about 10 minutes to get a mechanism picture from the mechanism library on the screen on a UNIX mini-computer. As earlier analyzed in Section 3.3 of Chapter 3 this is mainly attributed to a lack of the object relativity principle at both the conceptual database design level and the implementation level. This problem is recognized in developing LINKDES. An alternative database schema for the mechanism library is defined as shown in Fig. 9.27. Several remarks are given below:

- Attribute 'Mech-file' is defined as type of GMM. The value of attribute 'Mech-file' is simple a database file of the instantiated GMM. In concept this can be viewed as an exhaustive exercise of the object relativity principle, i.e. an attribute could be viewed as a database schema. The solution described in Fig. 9.27 can be regarded as an analogy to the actual library and the book for better understanding.

<table>
<thead>
<tr>
<th>Delft-University-Library</th>
<th>TADSOL-Mechanism-Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>Books</td>
<td>Mechanisms</td>
</tr>
<tr>
<td>Structure of books</td>
<td>Structure of mechanisms</td>
</tr>
<tr>
<td>Generic Book Model (GBM)</td>
<td>Generic Mechanism Model (GMM)</td>
</tr>
</tbody>
</table>

- Attribute 'Parameter-description' will describe the meaning of the parameters of the mechanism stored in the mechanism library for TADSOL, in particular the relationship of those parameters with the kinematic parameters of the mechanism.
Attribute 'Topology-code' is defined as a set-valued attribute to hold the unique codes of the mechanism topology related to different levels of the mechanism topology definition which were described in Chapter 6 in detail.

Figure 9.27. Data Representation of Mechanism Library for TADSOL.

The new database model for the mechanism library has been proved to be very efficient in retrieval of a mechanism from the mechanism library. With this model it takes about 2 seconds on a 386 SX (20-MHz) personal computer for the same work done by Kaper (1987).

Coupling of GMM and design results of TADSOL
The result of TADSOL, for each selected mechanism, is a set of parameters which may not be design parameters but the ratio of design parameters. As implied in the previous discussion of this chapter, it may be understood that GMM is always needed to retrieve a mechanism picture on the screen (each mechanism in the library will be stored in the computer with GMM-format). This means that it is necessary to establish the communication between TADSOL-parameters and GMM. The general scheme for this communication is shown below:

TADSOL-parameter -> Design Parameter Model (DPM) of LINKDES <-> GMM.

LINKDES-DPM is very similar to DSYNOP-DPM; only some attributes are dropped from DSYNOP-DPM, e.g. attribute 'Lower-boundary', etc. The above scheme implies an idea. It is envisaged that a mechanism found in the LINKDES environment should be allowed for further processing. This may include the following:

- change of certain kinematic parameters of this mechanism which may be done with EDMECH,
- adding more machine objects or putting this mechanism in the machine context, i.e. to connect with other mechanisms in a machine,
- adding mass so that the dynamic simulation can be carried out, and
- further dimension synthesis with, e.g. optimization with DSYNOP.
Automating adjusting process
Because the type of a mechanism found by TADSOL is already known, the nature of the adjusting process is equivalent to re-configure mechanism in the machine environment with
a set of new kinematic parameters. This is actually a kinematic analysis problem. The key
demand is therefore to automatically generate a kinematic model from GMM. This has been
solved already, as discussed in Chapter 8.

**Perspective**

Fig.9.28 shows several windows which are envisaged in LINKDES. In *Window-1* the
designer is describing the machine context, i.e. two ground nodes and motion directions of
the input and output. The required behaviour function is described in EDFUNC similar to that
for CAMDES. In *Window-2*, one of the results found by TADSOL is displayed. At that
moment, the mechanism is not configured at all. In *Window-3* the mechanism is configured
in the mechanism library context and in *Window-4* the mechanism is configured in the
machine environment. In *Window-2* till *Window-4* three buttons are necessary. Button
'Browser' can be used to look at the mechanism if there is more than one solution. If so,
button 'Select' can be used to choose the one which is most liked. Button 'Exit' offers an
opportunity for the designer to change his previous design decisions. *Window-5* symbolizes
the further design processing.

**Difference from the related work**

Leenders (1991) reported the integration of TADSOL into a CAD-environment called
Unigraphics II. The main difference of his work from the present study is that in his system
each mechanism in the mechanism library corresponds to a program routine responsible for
the kinematic analysis of this mechanism and thus drawing the mechanism picture. While in
LINKDES each mechanism corresponds to a database file of GMM-format. In the viewpoint
of the knowledge representation in the computer, his method is of procedural manner while
the method taken in LINKDES is of declarative manner (of data modelling in the context of
the present study). The consequence of this difference is that his system does not support the
further processing such as changing the structure of the mechanism found by TADSOL and
putting this mechanism into the machine context, *e.g.* to connect with other existing
mechanisms in a machine, for the further processing such as the dynamic simulation of the
whole machine (not just this single machine). This consequence is because in his system the
change of the mechanism structure is equivalent to the change of the program source code.
While in LINKDES the change of the mechanism structure is equivalent to the change of a
data file, or in the sense of data modelling, the change of the instance of a database schema,
i.e. GMM.

**9.5 Discussion and Conclusions**

The CIMOME architecture with its underlying design principles has been made more explicit
through the discussion of the implementation of three design synthesis tools environments in
this chapter. CAMDES integrates the tools such as TADSOCC, RUNMEC, GMMFEM,
FEMGMM, EDFUNC and EDMECH. DSYNOP integrates the tools OPLAM, EDMECH, EDFUNC, etc. and LINKDES integrates the tools TADSOL, EDMECH, EDFUNC, etc.

CAMDES is taken as a generic implementation pattern and therefore discussed in more detail. Its data communication scheme is exhibited, which in some sense complements the discussion of the CIMOME architecture in Chapter 2. The implementation practice has shown that the CIMOME architecture with its design principles results in a more flexible and open system and establishes a context for the concurrent system development. GMM, which has been established in Chapter 5, is further verified that it can be readily extended to represent the semantics of the cam mechanism design. In the discussion of the implementation of CAMDES, a global communication among the tools (sub-environments) and CIMOME top is considered by the establishment of a so-called project database, which is global to all design synthesis tools and CIMOME top. The database schema of the project database and the method of such communication are described. Results of the implementation are reported with several windows of CAMDES. A very remarkable implementation is about the design process management in a CAD/CAM environment. Following the process model proposed in Chapter 4, the implementation scheme considered and the result achieved in CAMDES are shown.

Following the concept of object and object view elaborated in Chapter 4, each design synthesis tool works on a certain view of the mechanism structure. That is reflected in the so-called Design Parameter Model (DPM) and its relationship with GMM. Several windows of CAMDES and DSYNOP are exhibited to show the result of the implementation owing to this idea. This idea is actually quite geared to the idea earlier proposed in Chapter 7, that is, a classification of Editors into a generic part and a specialized part. In the context of the discussion of the current chapter, those specialized parts of editors are built in individual design synthesis tools environments. The idea is further applied to the relationship between GFM (Generic Function Model described in Chapter 4) and some extra information for a required behaviour that DSYNOP concerns. In GFM, with its interface system EDFUNC, only common information about a required behaviour function is described, while extra information dependent on individual tool systems is assigned to individual tool systems. The result of the implementation of DSYNOP, following this idea, is exhibited.

From the results which have been achieved so far, it can be seen that design tools can be accessed fast and transparently and design process can be run in a flexible and intuitive way. To the author's knowledge, no published cam mechanism design system reaches the level of the design automation and the user-friendliness of CAMDES. The unique implementation of the computer-supported design process management adds a lot of values to the intelligence of the system. DSYNOP and LINKDES appear very promising in their design automation and user-friendliness, although the further implementation work remains to be done.
Concluding Remarks and Future Work

10.1 Introduction

The research described in this dissertation is aimed at developing a software environment (CIMOME) to integrate computer-based tools for designing and manufacturing mechanical systems (mechanisms in particular). The tools include those developed at the Laboratory of Mechanization of Production in the past two decades and those commercially available. The function spectrum of the tools covers the whole life cycle of the machine development. Both theoretical study and implementation have been carried out. The theoretical study has resulted in new observations on and approaches to modern CAD/CAM systems, which have been verified through the implementation of a prototype CIMOME ( Chapters 6 to 9). A very close coupling of software engineering and mechanical engineering has been demonstrated, which leads to a general concluding remark that it is critical to apply the knowledge of software engineering in improving the intelligence of application CAD/CAM systems. In this last chapter, a number of conclusions already drawn in the preceding chapters and some new ones are brought together with some remarks (Section 10.2). Future work is also identified (Section 10.3).

10.2 Concluding Remarks

Integrated environment architecture design

In comparison with a non-integrated system, an integrated system is more complex. It requires higher architectural skill to create an integrated system. The proposed design principles for environment architecture (Chapter 2) are thus significant as they help to make a better architecture for the environment. A large environment for tool integration should recognize the existence of individual tools - one of many features of an integrated
environment. Another feature is that an environment should at least contain such system components as a development process manager, a user-interface manager and a data & knowledge manager. The information relativity principle is applied so that a large environment is composed of sub-environments and certain tools (in the context of this work, design synthesis tools) are viewed as environments. This study has proposed the concept called horizontal integration and vertical integration which makes CIMOME general, open and, at the same time, implementable. The above ideas have been proved to be very favourable for decreasing the overall complexity of a system and for the concurrent system development.

Tools, as members of an integrated environment, must share something that can be generally concluded in three aspects: data, process and presentation. This is further conceptualized as a so-called three dimensional integration model or framework (Chapter 2). This model is very useful to evaluate an integration scheme with respect to cost, effort and benefit. Of course, the more sharing, the better in the sense of the functionality. In the implementation of CIMOME it is found possible to uniformly consider all these three aspects on tools that are developed in the past at our Laboratory. The presentation of implemented tools environments is quite similar. This will improve the user-computer interaction. This critical consideration has led to the decision to develop a general user interface management system (Chapter 7).

At the implementation level the data communication scheme among tools was defined first. The five data integration guidelines proposed (Chapter 2) are useful to make such a scheme rationally. The main criterion should be to reduce the complexity and maintain the autonomy of each individual tool, provided that the functionality of the data communication is not lost. The criterion is also applicable to the definition of the tool-tool triggering. Generally, for a large environment like CIMOME it is impossible to use simply the procedure manner for the tool-tool triggering because of the complexity of the system maintenance and the diversities of media on which tools are built. The proposed concepts (Chapter 2), such as the classification of tools into design tools and service tools and identification of data link processes, are therefore useful in determining the type of the tool-tool triggering interface. When the type of the tool-tool triggering is chosen via the program or process manner, an external protocol for the command message passing among different tool programs may have to be defined. Although some operating systems provide such a protocol, it may still be necessary to define such a protocol at a level higher for the sake of better portability. Such a protocol for the current CIMOME can be found in Chapter 2 and has been experimented in the implementation of CAMDES (Chapter 9).

It is not important to argue whether a CAD system must be process-oriented or product-oriented. The modern environments for CAD/CAM integration should equally emphasize both, as CIMOME does. Furthermore the object-orientation approach to the system development should not be confused that the system is product-oriented.
Product modelling and its implications to CIMOME

The increase of intelligence of CAD systems for the mechanism design has long been hindered because some important knowledge of software engineering is neglected. Examples are the data and knowledge representation of engineering objects or data modelling in the context of this work. The product modelling, a method used by designing CAD/CAM systems, can be considered as a kind of remedy. In the viewpoint of the system developer the product modelling is nothing more than to extensively represent the product information in the computer explicitly. In the viewpoint that a design is a mapping process from the function space to the attribute space, the minimal content of a complete CAD/CAM system should include a requirement model, an artifact model and a process model. Both quality and quantity of the information represented in each of these three categories of model, however, depend on a specific application and the system development stage. In this study of CIMOME, a behaviour function model (GFM for short, Chapter 4), a conceptual mechanism structure model (GMM for short, Chapter 5) and a design process model (Chapter 4) have been developed and implemented. It must be made clear that for the end-user (product developer) these models are still too implicit. This implies that user-interfaces must be built upon these models. In CIMOME, EDMECH is an interface for GMM (Chapter 7), EDFUNC is an interface for GFM (Chapter 4), and the user interface for the design process model has been exercised in the implementation of a sub-environment CAMDES for the compound cam design (Chapter 9). It is predicable that the computer-aided development process management will add values to intelligence of the modern CAD/CAM system.

On the issue of product model, most of the work in the literature enjoys establishing general framework including the whole life cycle of products: from marketing to product disposing. By contrast, the framework of a product model described in Chapter 4 of this dissertation starts to think in a different dimension or view to look at the product model. A product model is imagined as a kind of information sets in which elements are information blocks and related to one another like a network. With the objective to describe such a network irrespective of types of products, a notion called object and object-view is proposed. Two generic links exist among objects and object-views, that is, view-of and specialized-of. By applying the information relativity principle, an object-view can be object in another situation where some information blocks may be views of it. On each link between objects or object-views some functionality information and processes to maintain this link can be attached. The framework could be extended to develop a formal method to specify references among product data, which is critical to the change notification and propagation in an integrated system. Applications of the above idea in the implementation of CIMOME can be found in two places of this dissertation. The first place is the description of the relationship between GMM and the product assembly model at the embodiment design stage (Chapter 5). The product assembly model is considered as specialized-of GMM. The second place is that the design parameter model behind each design synthesis tool is considered to be view-of GMM (Chapter 9). An obvious benefit of doing so is the possibility to define a general system
component to maintain all those links to cope with the change notification and propagation.

GMM, a key to support the conceptual design of mechanisms
To well support the conceptual design of mechanisms it is necessary to explicitly represent the information of the conceptual mechanism structures in the computer. The information regarding the conceptual mechanism structure is what is represented in the Kinematic Mechanism Diagram (KMD for short) with possible attached comments from the designer, which might be called augmented KMD. (The augmented KMD belongs to the category of the engineering drawing.) As a common sense knowledge, an ideal situation for a better communication between two parties is such that the information to be exchanged is represented for both parties explicitly. Explicit representation in the sense of the computer is meant for the database representation, while explicit representation in the sense of the designer is meant for the engineering drawing representation. Because GMM is based on the augmented KMD and is a database model (Chapter 5), the information of the conceptual mechanism structure is made explicit towards both parties: the designer and the computer.

The above theoretical ideas have been sufficiently verified with the discussion of several implementations in Chapters 7-9. Two paradigms (analysis paradigm and synthesis paradigm) have been demonstrated and predicted. The analysis paradigm is referred to the sub-environment KINANA (Chapter 8) and the synthesis paradigm is referred to the sub-environments CAMDES, DSYNOP and LINKDES (Chapter 9). The CIMOME paradigm is therefore composed of these two paradigms. As it can be found, all these sub-environments are related to GMM.

Mechanism identification in the design process
It has been recognized that the mechanism identification should consider the evolution of a design process (Chapter 6). The implication is that the meaning of the mechanism topology may not be fixed on one level. Four levels of mechanism topology are defined. Furthermore the mechanism identification should not be limited to the kinematic chain identification; instead, it should include the function generator and path generator. Based on the study of the optimal code method in the literature for the kinematic chain identification, a classification has been found of code methods: the constrained optimal code and the unconstrained optimal code methods. It was found that the constrained optimal code method is more effective. Extension to the incident degree code method has led to a new approach. The new approach is based on the so-called vertex feature degree proposed in this study (Chapter 6). The vertex feature degree tries to extract more features of a vertex of a graph (the vertex of a graph corresponds to the body of a mechanism) and digitizes those features so that they can be given a sense of magnitude. Permutations to get an optimal code of a graph are subject to a predefined order of vertices in accordance with the magnitude of their vertex feature degrees. The algorithm with the computer program has been developed and tested on kinematic chains up to 28 bodies. This new approach works for the kinematic chain, the mechanism, the
function generator and the path generator with different levels of topology definition.

**Data or database modelling approach to CIMOME - a unique methodology**

One of the important approaches throughout the development of CIMOME might be called data or database modelling approach (Chapter 3). An important feature of this approach is not only to use the data modelling to develop the conceptual data & knowledge representation of engineering objects, but also to use a main memory DBMS system called ABC, which has been continuously improved in parallel with implementation of CIMOME, as a general tool to construct and manage complex data structures. Several benefits are achieved by doing so.

- The conceptual system representation can be highly portable because the conceptual data modelling approach used here (Chapter 3) is irrespective of any programming language and operating system.
- Engineering objects can be represented more naturally. As an informal argument of this point, one can find the similarity between our human society and a generic feature of most conceptual data modelling approaches, that is, *hierarchical*. (Most semantic database modelling approaches actually support two dimension hierarchical modelling.)
- The drawback of programming language FORTRAN, a scanty of general means to construct and manage complex data structures, is considerably alleviated due to the used approach with the supporting system ABC.

In this study, insufficient support for the conceptual data modelling with those contemporary semantic data modelling notions and approaches is also identified, for example, the notion of the complex object and its representation, the notion of object relativity principle, etc. The concept for the solution is introduced (Chapter 3). Furthermore it has been found that a main memory DBMS for interactive engineering applications is necessary. In the future, data modelling will play more and more an important role in the program system development, in spite of presence of the object-oriented programming languages. Two kinds of evidence have been found in the literature. One is by Van der Weerd (1992) who proposed to use database abstractions in programming with SmallTalk. The other is by Sim (1993) who concluded that bad object-oriented programming causes more problems than bad procedural programming. This implies that a considerable amount of experience is needed to master the object-oriented programming language.

Using object-oriented data modelling notion and approach does not mean that a good model of engineering objects can be guaranteed. To thoroughly understand the semantics of a specific engineering application, *e.g.* machine design, is far more important than to use a specific data modelling approach. The data modelling is like a hammer; how to effectively use the hammer is totally another question. A very simple but instructive example is that a lathe operator manufactures a workpiece. Different operators would produce very different quality (*e.g.* surface quality) for the same workpiece with the same lathe. Guidelines for the
conceptual data modelling proposed in Chapter 3 are therefore helpful and constructive.

**Features of intelligent CAD/CAM systems**
One of the aspects of intelligence of a CAD/CAM system is that the end-user is not confronted with irrelevant questions. Determination of which questions are irrelevant depends on the analysis of the roles played by and the knowledge required on the end-user at a specific design state. Consider the situation that the end-user who is performing the task: designing by analysis or simulation. The system that does not require the end-user to hold the theory or formulation of the motion equation of mechanisms (KINANA sub-environment, see Chapter 8) is certainly more intelligent than the system that more or less does, e.g. the commercial systems SAM (Delft-FEM theory), CAMLINK (Assur group theory), OSMEC (Assur group theory), etc. Those theories or formulations of the motion equation of mechanisms are therefore considered as irrelevant. For the end-user who is in the position to perform the stress and strain analysis, the knowledge that the increase of the number of elements gets more accurate result would be required. The basic concept of the finite element theory for stress and strain analysis is therefore relevant. To avoid irrelevant questions to the end-user, an integrated CAD/CAM system must be organized in a hierarchical manner. The hierarchy could be extended with more than one dimension. An example can be found in Chapter 7 where the organization of editor systems in the context of CIMOME is shown.

The model-based reasoning is another feature of intelligent CAD/CAM systems. This is true because on one hand the structure information of a designed artifact is now represented explicitly in the computer, and on the other hand designing can be viewed as a reasoning process and the results of reasoning are further affected on the structure of a designed artifact. In this way a 'lively picture' of model-based reasoning is naturally formed. Automating the finite element modelling described in Chapter 8 is a nice exercise of this feature.

**10.3 Future Work**

The computer aided design process management is an interesting issue worth researching. It is very helpful to show a design process explicit to the end-user. To support an iterative design process in the context of the computer aided design process management, the key point is the management of design alternatives and the design version. It is envisaged that recording of design states each time when design state variables are changed would be a solution to keeping an actual design path. When a design path is kept in a database and made persistent, reusing of the design process for similar or nearly similar design problems becomes possible. It has been observed that the reuse has already become an interesting issue in software engineering (Li, 1993). By making an analogy between computer program systems (products of software engineering) and machines (products of mechanical engineering), some
achievements might be taken from software engineering.

The link between GMM and assembly models of volumetric objects is worth further researching. The objective can be described. The system can allow the designer to draw a sketch of machine objects at the conceptual design site and/or the embodiment design site. No matter what top-down or bottom-up design process actually takes place, the link to those information blocks at these two sites is created and maintained with an actual design process. After that, any change of design either at the conceptual design site and the embodiment design site would be automatically managed. Some auxiliary diagrams, mainly to depict the feature of connections among application objects, such as the conceptual structure diagram (Sowa, 1984; Salomons et al., 1993) and the function means tree (Bracewell & Sharpe, 1994) would be helpful.

The CIMOME prototype should be further polished and made commercially available. To be a commercial system it is recommended to investigate and consider the implementation of the current prototype under a windowing system. This is very important for the future as a lesson has been learned from the implementation of CIMOME, that a considerable amount of time has been spent on developing certain fundamental tools such as the graphical system, the DBMS and the UIMS. It is also found that due to a lack of sufficient resources, the implementation of new ideas would be hindered.

The CIMOME prototype should at the same time be extended regarding its functionality. For example, it is most likely to integrate tools for the flexible dynamic simulation, the cam life cycle calculation, etc. Drivers must be included in GMM so that CIMOME is evolved to support a more complete machine design. GMM should be extended to spatial mechanisms. The behaviour function model should be extended to include more information about requirements for the machine design. A kind of knowledge base is expected which contains physical building blocks for machines for the reasoning from the function space to the attribute space. The work performed along the track of the so-called function modelling is interesting to build such a knowledge base and reasoning.
Reference


Ambekar A.G. and Agrawal V.P., 1987b.


Berg van der C.B., 1981.

Berge, P. van den, 1992.


A computer aided methodology for the development of conceptual schemes for mixed energy-transforming and real-time information systems, in *Computer aided conceptual design*, J. Sharpe and V. Ho (eds), LooseLeaf Co., U.K., 79-94.


Feature modelling and conversion--key concepts to concurrent engineering, *Computers in Industry* 21, 61-86.


Codd E.F., 1970.

A relational model of data for large shared data banks, comm. *ACM* 13(6), 377-387.


Date C.J., 1981.

*An introduction to database system*, Addison-Wesley.


Denavit, J. and Hartenberg, R.S., 1955.


Ways to smarter CAD-systems-examplied through design-concurrent calculation, *ICED '91*, 609-617.


Automated cam-mechanism synthesis and analysis, DE-Vol.46, Mechanical Design and Synthesis, ASME, 211-224.


ROSE and CHIDE: user interface management system implementation as object-oriented database system application, *Computer-aided design*, 8(22), 480-488.

*Database analysis and design*, Science Research Association, Inc.

*Introduction to system analysis and design*, Prentice Hall.


Henson B.W. et al., 1994.
Towards an integrated representation of function, behaviour and form, in *Computer aided conceptual design*, J. Sharpe and V.Ho (eds), LooseLeaf Co., U.K., 95-112.


*Concurrent design and analysis of mechanisms*, Rasna Corporation, California, USA.

Hua D., 1983.
Synthesis of the link-cam differential mechanism in a rapier loom, *Proc of 6th World Congress on TMM*, New Delhi, India.


The embodiment design for planar mechanisms: notions and implementations, Technical Report, BM-3.93, Laboratory for Mechanization of Production, Delft University of Technology.


239
ISO 3952/1-1981 (E/F/R).
Kinematic diagrams—Graphical symbols.
Development of new methodology for mechanical design by symbolized elements with movement, Annals of the CIRP Vol 41/1, 161-164.
An efficient algorithm for finding optimum code under the condition of incident degree, DE-Vol.47, Flexible Mechanisms, Dynamics and Analysis, ASME, 431-436.
Design of a data structure and a program for the automatic generation of the kinematic sketch for a designed mechanism, Master thesis, the TU Delft.
Kitajima Katsuhiro and Hiroyuki Yoshikawa, 1984
HIMADES-1: A Hierarchical machine design system based on the structure model for a machine, J. CAD, 16 (6), 299-307.
Expert system technology: development and application, Yourdon Press, A Prentice-Hall Company.
Object Identity, OOPSLA '86 Proceedings, 406-416.
A study on product modelling for integration of CAD/CAM, Integration of CAD/CAM.


Product modelling and 'information-integrated' engineering systems, Annals of the CIRP Vol.41/1, 201-204.


The disc cam regarded as a special finite element in kinematics, Proceedings of the sixth world congress of IFToMM, 1217-1220.


Kinematic optimization of mechanisms, a finite element approach, Dissertation, TU Delft.


Die Inselstruktur der Kinematik-Rechenprogramme--Brauchen wir einen "Deltaplan"?, VDI Berichte Nr.736,1-19.


OPLAM user manual, Lab of Mechanization of Production, Delft University of Technology, The Netherlands.


Conceptual design of mechanisms based on computational synthesis and simulation of

An Intelligent Mixed Symbolic/Numerical System for the Design of Mechanical Linkage,
Intelligent CAD II, H. Yoshikawa and T. Holden (eds), Elsevier Science Publishers B.V.
(North-Holland), 213-223.

Commonsense Reasoning about causality: deriving behaviour from structure, Artificial

From classic expert systems to models: introduction to a methodology for building model-
based systems, Topics in Expert system, G.Guida and C. Tasso (ed.), Elsevier Science
Publishers B.V. (North-Holland).

Development of a graphic interactive system with intelligent database for existing FEM
Netherlands.

A hierarchical data structure for representing assemblies: part 1, CAD, 17(1).

Computer aided design of plane link mechanism: integration of calculation and drawing


Model spaces and their use in mechanism simulation, Proc Insn Mech Engrs Vol.203,
167-174.
Li H.K., 1993.

RITL: An information system for application reuse-in-the-large, Ph.D dissertation, Faculty
of Informatics, Delft University of Technology, The Netherlands.

The product structure--the backbone of CIM, CIRP 41(1), 165-168.

A formal approach to structure, algebra & communication of complex objects, Ph.D
dissertation, Tilburg University, The Netherlands.

Database requirements of engineering applications an analysis, Internal Report Nr. 12/85,
University of Karlsruhe, Germany.

Databases for knowledge bases: empirical study of a knowledge base management system
for a semantic network, Data & Knowledge Engineering 7, 115-154.

Complex objects and their use in design transactions, IEEE database week, San-Jose, 115-
121.
Identification on spatial kinematic chains using incident degree and incident degree code, *Eight World Congress on the theory of machines and mechanisms*, Prague, Czechoslovakia, 999-1002.

Computer aided conceptual design of energy-transforming technical systems based on technical systems theory and bond graphs, in *Computer aided conceptual design*, J. Sharpe and V. Ho (eds), LooseLeaf Co., U.K., 59-78.


*Systematic approach to the design of technical systems and products*, VDI-2221, Design Handbook.


Computer aided design of mechanisms: the CADOM project of the Delft University of
TADSOL-Type and Dimension synthesis of link mechanisms, Proc. of the Symp. on Computer-aided design in mechanical engineering, Milano, 1976.
Spacebar: Kinematic design by computer graphics, Computer-Aided Design, 25(11), 727-735.
A Computer support tool for re-design - a prototype system resulting from applying a methodic design approach, Proc. of Int. Conf. on Engineering Design (ICED'93) Vol.3, The Hague, 1559-1569.
Motion simulation and interference detection of mechanisms, Master Thesis, Laboratory of Mechanization of Production, Delft University of Technology, The Netherlands.
A generalized symbolic notation for mechanisms, J. Eng. Ind. Trans. ASME.
Object-oriented systems analysis: modelling the world in data, Yourdon Press Computing
Series.
*Tool and database interfacing based on virtual objects*, Ph.D dissertation, Faculty of
Electrical Engineering, Delft University of Technology, The Netherlands.
Database abstraction: Aggregation and Generalization, *ACM Transactions on Database
system*, 2(2), 105-133.
The personal computer as an optimization tool for the synthesis of function generating
planar four-bar linkages, in *Computers in Engineering--1988 Vol 2*, V.A. Tipnis et al.
(eds), ASME, 73-78.
Sohn W.J. and Freudenstein, F., 1986.
An application of dual graphs to the automatic generation to the kinematic structures of
Sowa J.F., 1984,
*Conceptual Structures*, information processing in mind and machine, The Systems
Programming Series,
Toward a unified representation of mechanical assemblies, *Engineering with computers*,
Vol.6, 103-112.
*Adaptation and extension of the graphical interface package DIMECH to edit mechanisms*,
Master thesis, Lab of Mechanization of Production, Technical University of Delft, The
Netherlands.
Relational database design based on the Entity-relationship model, *Data & Knowledge
Engineering* 7, 47-83.
The degree code--a new mechanism identifier, *Trends and developments in mechanisms,
Ter Bekke, J.H., 1992b.
Complex values in databases, *Proc. of Int. Conf. on Data and Knowledge Systems for
Manufacturing and Engineering*, May, Hong Kong, 449-455.
Definitions of Tool Integration for Environments, *IEEE software*, IEEE Computer Society
press, March.


The concept of intelligent integrated interactive CAD system, Report CS-R8717, Center for mathematics and computer science, Amsterdam, The Netherlands.

Tomiyama T., ten Hagen P.J.W., 1987b.


Weerd van der P., 1992


Werff K. van der, 1975.

Kinematic and dynamic analysis of mechanisms: a finite element approach, Ph.D thesis, the Technical University of Delft.

Werff van der K., 1983a.
*SPACAR USER MANUAL a program system for the analysis of the motion of spatial mechanisms*, TU Delft.

Werff van der K., 1983b.

Heidelberg, 382-399.
Werff van der K., 1986.


A generic mechanism model for computer-aided conceptual design of machines, *Design Theory and Methodology -DTM'94-, USA*, 191-204.


*Object-oriented software*, Addison-Wesley Publishing Company, Inc.


Some discussions on the kinematic analysis of mechanisms by finite element method with the software "planar" and "spacer", Technical Report BM-2.90, Laboratory of Production Automation and Mechanization, the Delft University of Technology, The Netherlands.


Zhang W. and Werff van der K., 1993a.


Zhang W. and Werff van der K., 1993c.
Finite Element Modelling Automation for Mechanism Design and Simulation---Computer System Architecture and Finite Element Mechanism Model, presented in *9th Int. Conf. on CAD/CAM Robotics and Factories of Future*, New Jersey, USA.

Zhang W. and Zhang D., 1993d.

Zhang W., 1993e.

Zhang W., 1993f.
A Finite element mechanism (data) model for automating finite element modelling in mechanism design, Report Nr. BM-5.93, Laboratory of Mechanization of Production, Delft University of Technology, The Netherlands.

Zhang W., 1993g.
ABC (a simple virtual object data management system in FORTRAN language environment) User Manual (version 1), BM-2.93, the Laboratory of Mechanization of Production, Delft University of Technology.


Zhang W. and Dijk van A., 1994b.
On developing an intelligent editor system for machine conceptual design: application of database technology, in *Computer-Aided Conceptual Design*, J. Sharpe and V. Ho (eds), LooseLeaf Co., UK, 221-240.

Zhang W. and Werff K. van der, 1994c.

Zhang W., Crone H.A. and Werff van der K., 1994d.

The finite element method (3rd), McGraw-Hill.

Summary

Research and development for various computer applications in engineering are now being performed more and more actively in every industrial field. Computer Integrated Manufacture (CIM) technology is considered a key technology for manufacturing enterprises in the future. Integration is a means of achieving maximal profitability in an enterprise. It should be based on excellent communications among a team of product developers working on different product development stages. It is known that such communications can be facilitated by the computer-based integrated environment (environment for short).

The study described in this dissertation is concerned with developing such an environment, within which the designing and manufacturing activities of mechanical systems, mechanisms in particular, can be carried out in an integrated manner. Usually, human developers work with the help of computer-based tools. Therefore, the elements to be considered for the integration are these tools. The environment described is called Computer Integrated Manufacturing of Mechanisms Environment (CIMOME for short).

The significance of the tool integration, sometimes also termed CAD/CAM integration in the mechanical engineering domain, has been recognized by many researchers. However, on a particular object, the mechanisms in the present study, little progress has been made, especially in the research along the direction towards an implementation both economically justifiable and practically usable by many small enterprises. As a result, industries are gradually losing their interest in CAD of mechanisms.

It has been observed that the underlying concept for the tool integration usually follows a so-called ‘input-output’ pattern. The focus of this pattern is on: input data of tool A is made possible as output data of tool B. The major limitation of an integrated system based on this pattern is that relationship between input (and output) data and a designed object is seldom expressed explicitly in the computer. This limitation has led to the following consequences.

- Study on a natural representation of mechanisms into the computer system has been neglected. This inevitably produces the succeeding consequences as described below.
- Design automation has been hindered, because the task to interpret the input/output data and the relation to a designed mechanism must be done by human designers.
- Use of computer-based tools needs more underlying knowledge of the tools, such as the finite element approach to formulating the motion equation of mechanisms. This is so because of the discrepancy between the designer and the computer on the knowledge of the mechanism structure (topology and geometry). In the designer's world, the mechanism structure is what he draws on the sheet in a way of the engineering drawing, while in the computer's world the mechanism structure is represented with the underlying theory of
tools. A mapping from the mechanism in the designer's world to those represented in the computer's world has to be the responsibility of the designer - extra knowledge.

- Construction of a user friendly human-computer interface has essentially been hindered, because there is no common knowledge about a designed mechanism between the designer and these tools.

CIMOME established should improve the above situation. It is envisaged that the research should show the feasibility that the integration scope of CIMOME grows to cover the tools for the whole life cycle of machine and various development activities, albeit the current study and implementation may concentrate on the activities at the conceptual design stage. With the above objectives, the study generally consists of two parts: the theoretical part (Chapters 2 to 6) and the implementation part (Chapters 7-9). The division of the two parts may not be strict in the sense that some overlay is possible.

In the theoretical part, the basic ideas for the solution to CIMOME are described. It has been observed that the software environment for the software system development, researched in the software engineering, is quite similar to CIMOME in the context of the mechanical engineering. An analogy is made below.

<table>
<thead>
<tr>
<th>software engineering</th>
<th>&lt; = &gt;</th>
<th>mechanical engineering</th>
</tr>
</thead>
<tbody>
<tr>
<td>computer program</td>
<td>&lt; = &gt;</td>
<td>mechanical system or mechanism</td>
</tr>
<tr>
<td>tool (Editor, Linker, ...)</td>
<td>&lt; = &gt;</td>
<td>tool (Editor, Analysis, Synthesis)</td>
</tr>
<tr>
<td>CASE environment</td>
<td>&lt; = &gt;</td>
<td>CIMOME</td>
</tr>
</tbody>
</table>

| etc.                  |       |                        |

A big progress has been made in the software engineering, such as the environment reference model proposed and implemented by Buxton & Druffel (1984), in which the common database approach was used.

Based on the above observations, the general attitude to developing CIMOME is to learn some philosophy from the software engineering. This leads to a common database for mechanism structures and all other relevant information generated during a design process. Furthermore, the general approach to the CIMOME architecture development is not limited to the data integration via the common database - probably only one dimension in the entirety of the integration of tools. The CIMOME architecture also considers the support of the mechanism design process enabled by those tools (termed process integration) and the user-computer interaction (termed user-interface integration). The general strategy for implementation is to consider:

- the accessibility of small companies,
- the practical usability, and
- the minimal need for underlying knowledge of design tools.
The main methodology used in developing CIMOME is the data (or database) modelling approach and the object-oriented system development approach.

The theoretical study has generated the following results in particular.

- A database approach to the CIMOME realization is formulated. The approach has the following features.
  - In philosophy, the designing of data structures is replaced by the data modelling.
  - A set of notions with guidelines are used to construct conceptual data structures for application objects. The notions include those found in several well-known semantic data models and those further developed in this study such as the composite object data representation, the object relativity principle and the meta model.
  - At the implementation level, a memory database management system is developed and used.

This approach is used throughout the present study and proves to be very useful.

- A set of principles for designing an integrated environment architecture are established. The principles aim at reducing the overall complexity and facilitating the implementation and maintenance of an integrated environment. In particular, an architecture should allow an environment to be implemented concurrently and incrementally. According to these principles, the CIMOME architecture is formulated. An important feature of the CIMOME architecture, in the viewpoint of functionality, is to provide a context for the design process management, in addition to the product data management.

- Notions of product modelling with respect to the CIMOME development are clarified. A general framework of product model is established by following a new concept to structure the product information. This concept includes the classification of the product information into a so-called 'object and object-view' and the definition of two generic links between the objects and/or object-views, i.e. 'A view-of B' and 'A specialized-of B'. (Here A and B are objects and/or object-views.) The concept is well applied to formulate the relationship between a mechanism structure model (see below) and design synthesis tools.

- A partial requirement model behind CIMOME is formulated. This model mainly includes the information regarding the input-output behaviour function which is a very important aspect of the requirements for the mechanism development. A user-interface prototype based upon the model has been developed and has shown that the model is of user-orientation.

- A model to represent the conceptual mechanism structure information is formulated (termed GMM). It is argued that the so-call mechanism conceptual structure information must be what is usually represented in the kinematic mechanism diagram with possible designer's comments (the augmented kinematic mechanism diagram). It is analyzed and concluded that lack of such a model is an essential reason why the conceptual design of mechanisms is hindered. The model is defined as a generic model in the sense that
- the model is a collection of classes with defined types - a schema set,
- an individual mechanism is an instantiated model subject to the defined model schema,
- the model is easily extended because the model is based on a proposed object-based description of the mechanism conceptual structure and used semantic data modelling approaches.

A user-interface prototype based upon GMM has been developed and has shown that GMM is of user-orientation. Against the concept of the object and object-view mentioned above, design synthesis tools actually work in certain views of GMM (these views are also model). Therefore GMM has been extensively used in the implementation for bringing several design synthesis tools into CIMOME. GMM plays an essential role in CIMOME. For example, it has been shown that from GMM a finite element view model of mechanisms for the kinematic analysis or simulation can be automatically generated. It has also been shown that those design parameter models behind design synthesis tools enable to communicate with GMM. Similar work is exhaustively compared with GMM and its underlying ideas.

- A design process model is developed. The model is a collection of database schema and a particular design process is an instantiated model. In this way the model is general. Verification of the model has been conducted by implementing a sub-environment of CIMOME for the compound cam mechanism design. In this sub-environment the design process is made explicit to the designer.
- The role of the mechanism identification with a design process is recognized. The definition of the mechanism topology could be different with design stages. Four levels of the mechanism topology are proposed. A new method with algorithm is developed to carry out the mechanism identification for the kinematic chain, mechanism, function generator and path generator at various levels of mechanism topology. The new method is extended from a so-called incident degree code method in the literature.

The implementation is mainly to provide verifications of the theory described and shows the perspectives of CIMOME with regard to the goals that CIMOME is desired to achieve. The implementation produces about 15000 program statements and includes the following systems in particular.

- A main memory database management system. The system enhances programming language FORTRAN in constructing complex data structures. An important feature of the system, among others, is to allow for the dynamic evolution of the data type definition.
- A user-interface management system (UIMS for short). The system allows fast phototyping of an application graphical user interface. The control mode which is supported by the system is a shared control mode - both applications and UIMS enable to control the dialogue procedure between the user and the computer. The main ideas behind developing this system are to define an application user-interface in a declarative manner and to make a separation between the application-specific semantics management and the common user-interface management which can be handled by UIMS.
User-interfaces systems for the mechanism structure model and function requirement model.

A system to perform the automatic finite element modelling for the kinematic analysis and design. The system takes the mechanism structure model as input to generate output which is an input data file of a finite element package. The result generated by the finite element package can be further interpreted and converted to those representations known to the designer. In this way the finite element theory behind tools is hidden from the end-user (the designer).

Several design tools environments - sub-environments of CIMOME. The main ideas and procedures to implement these systems are described. The integration of CAD/CAM is demonstrated by the example that for a particular machine component cam, its NC-code can be generated after the designer is satisfied with the design result. Several practical design cases are exercised by using these systems to show the facilitating of design. Comparisons are also made with other known systems and the situation where an integrated environment is absent.

In conclusion, the study has demonstrated the feasibility of the improvement of product development via an integrated environment such as CIMOME. The principles, methods and techniques to develop such an environment have been developed. Although there is still some work to be done to lead CIMOME to a polished system for the practical use, the current implementation has shown that CIMOME has a bright perspective.
Samenvatting

Technische computer-toepassingen dringen door tot in iedere tak van industrie. De technologie die wordt aangeduid met Computer Integrated Manufacturing (CIM) wordt voor produktiebedrijven als de belangrijkste technologie voor de toekomst beschouwd. Integratie is een hulpmiddel om een onderneming maximaal winstgevend te maken. Integratie moet gebaseerd zijn op een zeer goede communicatie tussen de mensen die in de verschillende ontwerp- en produktiestadia aan een produkt werken. Een goede communicatie is mogelijk in een geïntegreerde, op computergebruik gebaseerde, werkomgeving.


Met betrekking tot mechanismen, waarop het voorliggende onderzoek betrekking heeft, is op dit gebied echter tot nu toe weinig vooruitgang geboekt. In het bijzonder is weinig onderzoek verricht naar de implementatie van geïntegreerde systemen die praktisch bruikbaar en economisch haalbaar zijn in kleine ondernemingen. Als gevolg daarvan verliezen bedrijven langzamerhand hun interesse in CAD van mechanismen. Veelal is het concept voor integratie gebaseerd op het zogenoemde "invoer-uitvoer" patroon. Dit patroon heeft de volgende kenmerken: Als invoer- en uitvoergegevens van gereedschap A kunnen de uitvoergegevens van gereedschap B gebruikt worden. De belangrijkste beperking van een systeem, waarbij integratie is gebaseerd op dit invoer-uitvoerpatroon, is dat de relatie tussen invoer- of uitvoergegevens en het betreffende ontwerp-object zelden expliciet in de computer wordt weergegeven. Deze beperking heeft de volgende consequenties:

- Onderzoek naar een natuurlijke wijze van representatie van mechanismen is niet verricht.
- De voortgang in automatisering van het ontwerpproces wordt belemmerd omdat het interpreteren van de invoer- en de uitvoergegevens en het vaststellen van de relatie met het ontworpen mechanisme aan de gebruiker/constructeur is overgelaten.
Voor het gebruik van computer-hulpmiddelen is meer kennis over het gebruikte gereedschap zelf nodig, zoals bijvoorbeeld van de eindige elementen methode om de beweging van een mechanisme te kunnen formuleren. De geringe vooruitgang in de automatisering op dit gebied wordt veroorzaakt door de discrepantie in kennis over mechanisme-structuren (topologie en geometrie) tussen de ontwerper/constructeur en de computer-systeemontwerper.

In de wereld van de constructeur/ontwerper wordt de structuur van een mechanisme weergegeven door een technische tekening. In de computerwereld wordt een mechanisme-structuur weergegeven op een manier die bepaald wordt door de theorie waarmee het gereedschap is ontwikkeld. De vertaling van het mechanisme uit de ontwerperswereld naar gegevens voor de computerwereld is de verantwoordelijkheid van de ontwerper/constructeur. Dit maakt extra kennis noodzakelijk.

Het ontwikkelen van een gebruikersvriendelijke mens-computer interface is een verwaarloosd gebied omdat er wenig gemeenschappelijke kennis over ontwerpen van mechanismen aanwezig is bij de ontwerper/constructeur en de ontwerper van de computer-hulpmiddelen.

CIMOME zal de hierboven weergegeven situatie kunnen verbeteren. Het verrichte onderzoek toont de haalbaarheid hiervan aan. Bij verdergaande integratie zal CIMOME het gehele gebied afdekken met gereedschap voor ontwerp, fabricage en gebruik (life cycle) van een machine en van de verschillende ontwerppactiviteiten.

Het verrichte onderzoek en de implementatie van het voorgestelde systeem is in hoofdzaak gericht op activiteiten in de conceptuele fase van het ontwerpproces.

Het onderzoek is globaal in twee delen verdeeld. Een theoretisch deel (hoofdstukken 2 t/m 6) en een deel implementatie (hoofdstukken 7 t/m 9).

In het theoretische deel zijn de ideeën beschreven die de basis vormen voor CIMOME. Daarbij is gebleken dat de werkomgeving voor het ontwikkelen van software systemen in de informatica veel overeenkomst vertoont met de werkomgeving van CIMOME in de werktuigbouwkunde.

Onderstaand zijn een aantal van deze analogieën weergegeven.

<table>
<thead>
<tr>
<th>Software engineering</th>
<th>&lt;=&gt;</th>
<th>Werktuigbouwkunde</th>
</tr>
</thead>
<tbody>
<tr>
<td>computerprogramma</td>
<td>&lt;=&gt;</td>
<td>mechanisch systeem (mechanisme)</td>
</tr>
<tr>
<td>gereedschap</td>
<td>&lt;=&gt;</td>
<td>programma's voor deel-vraagstukken (analyse, synthese)</td>
</tr>
<tr>
<td>(editor, linker)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>werkomgeving voor toepassingen</td>
<td>&lt;=&gt;</td>
<td>CIMOME</td>
</tr>
</tbody>
</table>

256
In de informatica is veel vooruitgang geboekt met het "environment reference model" dat door Buxton & Druffel (1984) is voorgesteld en ingevoerd op basis van theorieën over een gemeenschappelijke gegevensbasis (common database). Bij het ontwikkelen van CIMOME is gekeken naar de boven aangegeven analogie en is lering getrokken uit de in de informatica gangbare filosofie ten aanzien van het ontwikkelen van software systemen.

Dit heeft er toe geleid dat voor mechanisme-structuren en alle andere relevante informatie gedurende het ontwerpproces een common database is gebruikt. CIMOME kent drie integratielijnen. Ten eerste de integratie van data via de common database, ten tweede het ontwerpproces van mechanismen (proces-integratie) en ten derde de mens-computer interactie (user-interface integratie).

Voor implementatie moet rekening worden gehouden met:
- de toegankelijkheid voor kleine bedrijven,
- de praktische bruikbaarheid en
- de minimaal noodzakelijke kennis voor het gebruik van de hulpmiddelen.

De belangrijkste methode die bij het ontwikkelen van CIMOME is gebruikt is die van data (of database) modellering en de object-georiënteerde systeem benadering.

Het theoretisch onderzoek heeft vooral het volgende opgeleverd:
- Een database aanpak voor CIMOME met de volgende kenmerken.
  - Het ontwerpen van data-structuren is vervangen door het modelleren van data.
  - Een set begrippen met richtlijnen die wordt gebruikt voor het ontwerpen van conceptuele data-structuren voor objecten in een specifiek toepassingsgebied. Deze begrippen omvatten ook de begrippen die men vindt bij de verschillende bekende semantische data modellen en de begrippen die in dit onderzoek zijn ontwikkeld voor o.a. de representatie van samengestelde object data, de relatie tussen objecten en het meta model.
  - Op implementatie-niveau is een database management systeem gebruikt.

Deze aanpak is consequent in het gehele onderzoek met succes toegepast.

- Een set ontwerpregels voor het ontwerpen van de architectuur van de werkomgeving is vastgelegd. Deze ontwerpregels beogen de complexiteit van het geheel te verminderen en de implementatie en het onderhoud van geïntegreerde systemen te vergemakkelijken. In het bijzonder zal de architectuur het mogelijk moeten maken dat de verschillende aspecten van de werkomgeving gelijktijdig kunnen worden uitgebreid. De architectuur van de werkomgeving van CIMOM (CIMOME) is volgens deze ontwerpregels geformuleerd.

Een belangrijke eigenschap met betrekking tot de functionaliteit van deze werkomgeving is het bieden van een structuur voor het beheersen van het ontwerpproces, aanvullend op het beheersen van de produktgegevens.

- Het verduidelijken van begrippen voor produktmodellering met betrekking tot de ontwikkeling van de CIMOM-werkomgeving. Een algemene raamwerk voor een
produkt-model is opgezet volgens een nieuw concept voor het structureren van de produktinformatie. Dit concept bevat tevens de indeling van deze informatie in wat men noemt "object en object-view" en de definitie van de twee algemene verbanden hiertussen; A is "view" van B en A is verbijzondering van B (hierbij zijn A en B objecten en/of object-views). Het concept is goed toepasbaar bij het formuleren van de relatie tussen het structuurmodel van een mechanisme (zie onder) en de hulpmiddelen voor de synthese.

- Een model van eisen achter CIMOME is gedeeltelijk geformuleerd. Dit model bevat voornamelijk de informatie die betrekking heeft op het gedrag van de invoer-uitvoer functie. Dit is een heel belangrijk onderdeel van de eisen bij het ontwerpen van mechanismen. Een prototype voor een gebruiker-interface, die gebaseerd is op dit model is ontworpen. Aangetoond wordt dat het model gebruiker-georiënteerd is.

- Een algemeen mechanisme-model (GMM, Generic Mechanism Model) is gedefinieerd. Dit model geeft de structuur van het mechanisme-concept weer. Aangetoond wordt dat de informatie over een mechanisme in de conceptuele fase die informatie is die gewoonlijk wordt weergegeven in een kinematisch schema van het mechanisme, eventueel aangevuld met commentaar door de ontwerper (aangevuld kinematisch schema). Geconcludeerd wordt dat het ontbreken van een dergelijk model de wezenlijke oorzaak is voor de achterstand in het ontwikkelen van het conceptueel ontwerpen van mechanismen. Het model (GMM) is algemeen in die zin dat
  - het model een verzameling is van klassen van gedefinieerde typen (schema set),
  - een afzonderlijk mechanisme een van het gedefinieerde schema-model afgeleid exemplaar is.
  - het model gemakkelijk kan worden uitgebreid, omdat het model gebaseerd is op een object georiënteerde beschrijving van de structuur van het mechanisme-concept en het semantische data model wordt gebruikt.
Op basis van dit GMM-model is een prototype gebruiker-interface ontwikkeld, daarbij is gebleken dat het GMM-model gebruiker-georiënteerd is.

- Tegen de achtergrond van het eerder genoemde concept "object/object-view" opereren ontwerp-gereedschappen op views van het GGM-model (deze views zijn op zich weer een model). Het GMM-model is daarom uitgebreid toegepast bij het implementeren van de ontwerp-gereedschappen in CIMOME. Het GMM-model speelt zodoende een belangrijke rol in CIMOME. Zo wordt bijvoorbeeld aangetoond dat met behulp van GMM automatisch een eindig elementen model kan worden ontwikkeld voor de kinematische analyse of simulatie van een mechanisme. Met een dergelijk model voor de ontwerpparameters uit de synthese-gereedschappen is communiceren met het GMM-model mogelijk.

- Onderzoek op dat gebied is uitgebreid vergeleken met het GMM-model en de er achterliggende ideeën.

- Een model voor het ontwerp-proces is ontwikkeld. Dit model is een verzameling schema's van databases. Het model is algemeen in de zin dat een bepaald ontwerpproces daarin een exemplarisch model is.
Verificatie van het model heeft plaats gevonden door een sub-werkomgeving van CIMOME te implementeren voor het ontwerpen van een samengesteld nok-mechanisme. In deze sub-werkomgeving wordt het ontwerpproces expliciet gemaakt voor de ontwerper.

- Identificatie van mechanismen speelt een belangrijke rol bij het ontwerpproces. De definitie van de topologie van een mechanisme zou verschillend kunnen zijn in de verschillende stadia van het ontwerpen. Vier mogelijke niveau's worden voorgesteld voor de topologie. Er is een nieuwe berekeningsmethode ontwikkeld voor de identificatie van een mechanisme. Een mechanisme wordt daarbij beschouwd als kinematische keten, als mechanisme, als functie-generator of als baan-generator en wel op de verschillende topologie-niveau's. Deze berekeningsmethode is een uitbreiding van de in de literatuur als "incident degree code" aangeduide methode.

De implementatie heeft voornamelijk het doel de opgestelde theorie te verifiëren en aan te tonen dat CIMOME aan de gestelde verwachtingen voldoet. De implementatie resulteert in ca. 16.000 programma-regels en omvat o.a. de volgende systemen:

- Een User-Interface Management System (UIMS). Dit systeem maakt het mogelijk om in korte tijd een schermindeling te maken voor een grafische interface toepassing. De besturing die door het systeem wordt ondersteund is de zogenoemde "shared control mode". Voor beide, applicatie en UIM, is het daarbij mogelijk om de procedure voor de dialoog tussen mens en computer te sturen.

De belangrijkste ideeën achter het ontwikkelen van dit systeem zijn het definiëren van een gebruikers-interface voor een toepassing op een declaratieve manier en het maken van een scheidt tussen het toepassings-specifieke semantisch georiënteerde management en het management voor de gewone gebruiker-interface die door het UIMS kan worden bewerkt.

- Gebruiker-interface systemen voor het structuurmodel van het mechanisme en het model voor de functionele eisen.
- Een systeem om automatisch het eindige elementen model op te stellen voor de kinematische analyse en het ontwerp. Het systeem gebruikt het structuurmodel van het mechanisme om een invoer-data file te genereren voor een eindige elementen programma-pakket. De resultaten uit het eindige elementen pakket worden verder geïnterpreteerd en omgezet in een representatie-vorm die bekend is bij de ontwerper. Op deze wijze is de theorie van de eindige elementen methode verborgen gehouden voor de eind-gebruiker (in dit geval de ontwerper).
- Verschillende werkomgevingen voor hulpmiddelen bij het ontwerpen (sub-environments van CIMOME). Hiervan worden de belangrijkste ideeën en procedures voor het implementeren beschreven. De integratie van CAD/CAM wordt gedemonstreerd aan de
hand van een voorbeeld waarbij voor een bijzonder machine-element, een nokschijf, de NC-code wordt gegenereerd, echter niet dan nadat de ontwerper tevreden is met het resultaat. Verschillende praktijkgevallen worden bewerkt onder gebruikmaking van deze systemen om de kwaliteitsverbetering van het ontwerpen te tonen. Het systeem wordt vergeleken met andere bekende systemen en met een situatie waarbij er geen geïntegreerde werkomgeving aanwezig is.

Geconcludeerd kan worden dat het onderzoek de haalbaarheid heeft aangetoond van de verbetering van de produkt-ontwikkeling door middel van een geïntegreerde werkomgeving als CIMOME.

De principes, methoden en technieken om een dergelijke werkomgeving te ontwerpen zijn ontwikkeld. Toch zal er nog veel werk gedaan moeten worden om van CIMOME een goed lopend systeem te maken dat in de praktijk toepasbaar is. De huidige implementatie heeft aangetoond dat CIMOME tal van unieke eigenschappen en een veelbelovende toekomst heeft.
Appendix A

Definition of Concepts from Information Technology

Definition A.1. Data is a representation of facts, concepts, or instructions in a formalized manner suitable for communication, interpretation, or processing by human or automatic means (ANSI/IEEE, 1983). In developing CIMOME the word 'instructions' is sometimes taken away from the above statement reserved for the definition of the term 'knowledge representation' ('knowledge' for short), and therefore we have the definition 1.2 below.

Definition A.2. Knowledge is a representation of instructions in a formalized manner suitable for communication, interpretation, or processing by human or automatic means.

Definition A.3. Information is a recognized view of data and knowledge. Data encodes the information about what to be, while knowledge encodes the information about how to do.

Definition A.4. Object can be facts, concepts, or instructions (abstract object) or physical artifacts (physical object). It should be pointed out that the term information may sometimes be used as view upon the object, instead of data and/or knowledge.

Definition A.5. Context (of view) is the information (data and/or knowledge) which founds the current view on the current object.

![Diagram](image)

Figure A.1. Principles of Information Relativity

Definition A.6. Information relativity concerns the ability to view and manipulate information in the way most appropriate for the viewer under application domain-specific context. The notion of information relativity may be made analogy to 'special relativity principle' in physics, i.e. Principle of Relativity (Galilao): No experiment can measure the absolute
velocity of an observer (Schutz, 1985). For convenience of use, two principles are made below from the concept of information relativity:

**Principle I (of information relativity).**
Different information (or views) must be possible from the same object (Fig.A.1a).

**Principle II (of information relativity).**
View i's context (context i) can be a view i+1 in context i+1, and this may happen recursively (Fig.A.1b).

The driving source to conclude information relativity principle above comes from two:

- data and knowledge engineering, *e.g.* 'object relativity' (Smith & Smith, 1978) and 'semantic relativity' (Brodie, 1984) and 'meta knowledge' (Hayes-Roth et al, 1983), and
- engineering application, *e.g.* 'meta model' (Ter Bekke, 1992; Tomiyama & Hagen, 1987a).

The principle of information relativity can be a universal reference to clarify some ambiguities related to the terms such as, *object relativity* and *meta model* which are important in building larger data-intensive software system like CIMOME.

**Definition A.7. Semantics** is the relationships between symbols and their meanings. *Syntax* is the relationship among symbols, independent of their meanings or the manner of their interpretation and use (ANSI/IEEE, 1983). For example, the following is a syntax, which is also called assertion (ter Bekke, 1992):

\[
\text{type } O = A_1, A_2, A_3, \ldots, A_n. \tag{A1}
\]

If a semantic *rule* is added, say the symbol on the left which replaces O is called object and the symbols on the right are called properties of the object, then the following expression is given a meaning:

\[
\text{type gear } = \text{module, pressure_angle, number_of_teeth.} \tag{A2}
\]

That is, a gear has properties: module, pressure angle and number of teeth.

Furthermore the symbols that have been chosen on the right side are not arbitrary in the sense that a symbol, say *mouse* can not be put on the right side simultaneously with the symbol *gear* being on the left side. The criterion to exclude the *mouse* as a property of the gear is the knowledge in the human's mind. So the semantics is subjective in nature.

262
Curriculum Vitae

Born in Shanghai, China, on 8 March 1959, the author graduated and got B.Sc degree from the Department of Equipment and Machine for Process Industry, the East China University of Technology, Shanghai, China, in 1982. He got his M.Sc degree from the Department of Mechanical Engineering, the China Textile University, Shanghai, China, in 1984. He then worked for the China Textile University over the period from 1984 to 1990. In 1987, he was appointed as an assistant professor. He was, in 1990, a visiting research fellow at the Laboratory of Mechanization of Production, Department of Mechanical Engineering & Marine Technology, the Delft University of Technology, the Netherlands. In November 1990, he started with the research, at the same Laboratory, mainly on the CAD/CAM of mechanical systems, leading to the doctor's degree. This research was carried out under the supervision of Prof.ir. H.A. Crone, Prof.dr.ir. K. van der Werff and ir. A. van Dijk.