INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

Université d'Ottawa · University of Ottawa

Computer Automated and Integrated Design

(CAID)

Thesis submitted

to the School of Graduate Studies

in partial fulfillment of the requirements for the

degree of Doctor of Philosophy in

Mechanical Engineering

by

© Said Farahat

Ottawa-Carleton Institute for

Mechanical and Aeronautical Engineering

University of Ottawa

Ottawa, Ontario

Canada, K1N 6N5

Abstract

Design engineers now fully recognize the remarkable power of computers, and realize that they must fully optimize their use in the design process. The Computer Automated and Integrated Design (CAID) system is a valuable tool for this purpose. This system offers enhancement from other researched design systems in many ways. It is a shell for easily integrating the knowledge of different design processes. The hierarchical form of the knowledge stored in the system makes it easy for the knowledge engineer to expand it and easy for the designer to understand. The increased of access to its stored knowledge, allows the knowledge engineer to implement the process using the least possible number of codes.

This research develops the CAID system and demonstrates its advantages through the designing of a gear train in an interactive design process. This study develops a system which allows designers to use the experience of expert designers while not placing any restriction on the designer's capability to develop creative designs. It provides a creative environment that also supports an automated design capability. The system can provide all the necessary design tools by easily integrating new tools into the environment. The system is a flexible, interactive and integrated environment and is adaptable to various design fields. The designer is able to process more than one design case simultaneously and switch between various design cases and control their process.

Acknowledgements

I would like to express my appreciation to my supervisor, Dr. A. Fahim, for his support and guidance. I also wish to thank the members of the advisory committee, Dr. M. Munro and Dr. G. Kardos. A thanks also to Dr. P. Frise and Dr. J. Raymond for their comments.

I appreciated the proof reading of my thesis by Mr. D. Seaman, and assistance of Dr. Vahedi, Dr. Kochakzadeh and my son Hamid Farahat. I also thank the administrative staff in the Department of Mechanical Engineering at the University of Ottawa.

I would like to thank my family for their patients during my study period, especially my wife, Mrs. M. Vahedi Notash and my children . A special thanks to our families, my parents, sisters and brothers (and in-laws), for their supports.

I gratefully acknowledge the Government of Islamic Republic of IRAN for offering a post-graduate scholarship which makes this work possible

And thank God who makes everything possible.

Table of Contents

Chapter 3 : Object Oriented Systems

Chapter 4 : Computer Automated and Integrated Design Philosophy

Chapter 6 : CAID Environment

Chapter 7 : How CAID Works

Appendix A : The Object Modeling Technique (OMT)

List of Figures

Chapter 1

Introduction

1.1 Introduction

Although most computer systems used in the mechanical engineering design process are drafting and analysis tools, there has been growing interest in computers as knowledge experts. As knowledge grows rapidly, computers are being used to handle the expanding volume of sophisticated information. Engineering designers need an automated and integrated computer system which extracts this knowledge and makes it flexible, interactive and easily usable.

Current software, mostly designed by specialized engineers, deals with specific individual design areas, for example, VLSI (Very Large Scale Integration) circuit design. Analysis software is another type of application software related to design. Analysis software equips the designer with powerful tools to examine the feasibility of the design solutions. Finite elements analysis and solid modeling packages are examples of this type of software.

Although current design software has been of great assistance, it has some drawbacks. It has certain limitations and often does not quite meet the needs of designers. Available application software programs are not flexible enough to adjust to new design methods and to the latest technologies. The main problem with application software arises from a weakness in the area of integration. Designers lose time as they transfer the results of one package to another for analysis and then back again for a re-design.

Graphics were the most important feature of most computer aided design (CAD) research and development in the past years. The software typically involved two-dimensional drafting systems and three-dimensional wire-frame and solid modeling systems, along with imaging and presentation systems. Synthesis and simulation systems (e.g., Working Model which is a motion simulation software from Knowledge Revolution) represent a further interesting development in engineering design research. Unfortunately, however, there has been little work done to develop computer systems that deal with design as a whole, rather the work has focused on very specific design areas and, for the most part, has ignored the overall process.

1.2 Objectives of the Research

Unlike other research on mechanical engineering design which emphasizes the automation of the design process, this work not only considers that element but places more emphasis on the flexibility and expandability of the system. Almost all the previous software operates in a way that the user enters the input data and any restrictions necessary, then the software processes this information and arrives at the final solution, but there is no interaction with the designer during this process. Contrary to this, the Computer Automated and Integrated Design (CAID) system is a flexible approach which communicates with the user and follows his or her instructions all the way from the high level process design stages down to the details of a part design. Hence, the user has complete control over the design process and can guide the direction of the process although some of the design factors might remain as an automatic process.

The objective, in developing CAID, is to contribute a system which allows designers to draw

on the experience of experts. CAID does not put any restriction on the designer's capability to develop creative designs but presents a creative environment which also supports an automated design capability, although this might seem to be contradictory. The following abilities were proposed to guide the implementation of the CAID system [1]:

-to be a design shell or base on which the user builds;

-to provide all necessary design tools, or allow easy integration of tools into the environment;

-to be a flexible and integrated design environment;

-to have expendability and growth potential;

-to be adaptable to design in various fields;

-to be a design system which grows and changes with advances in technical knowledge;

-to have the design system knowledge organized in a modular building block fashion;

-to have a senior designer who contributes his design knowledge to the system and a junior designer who uses the system for design;

-to allow a designer the ability to better control the design process;

-to allow a designer ability to process more than one design case simultaneously;

-to enable a designer to switch between various designs cases;

-to be a facility to store design cases and because of their flexibility, use them as a starting point for future designs.

Many of these advantages of the CAID system are not achieved in the other mechanical engineering design systems developed to date. The purpose of this work is to develop a CAID system and demonstrate its advantages through the designing of a gear train in an interactive design process.

1.3 Related Works

Advanced, intelligent and expert systems have been recently developed because of the demand for more extensive computer-aided support in the engineering design process, due to the efforts of automating the design process. Most of these systems have been developed in fields other than Mechanical engineering. There are few systems which were developed specifically for mechanical engineering design. Five different published papers related to this research are considered in this section. They are discussed chronologically as follows:

-"A Computational Approach to Conceptual Design of Mechanical Systems" [2]

This research is concentrated on the development and computer implementation of a mechanical conceptual design model , called 'Design by Exploration', emphasizing on the process automation. The model has the claim that optimal designs may be obtained through a step-wise transformation of an initial functional description of a device to a structural

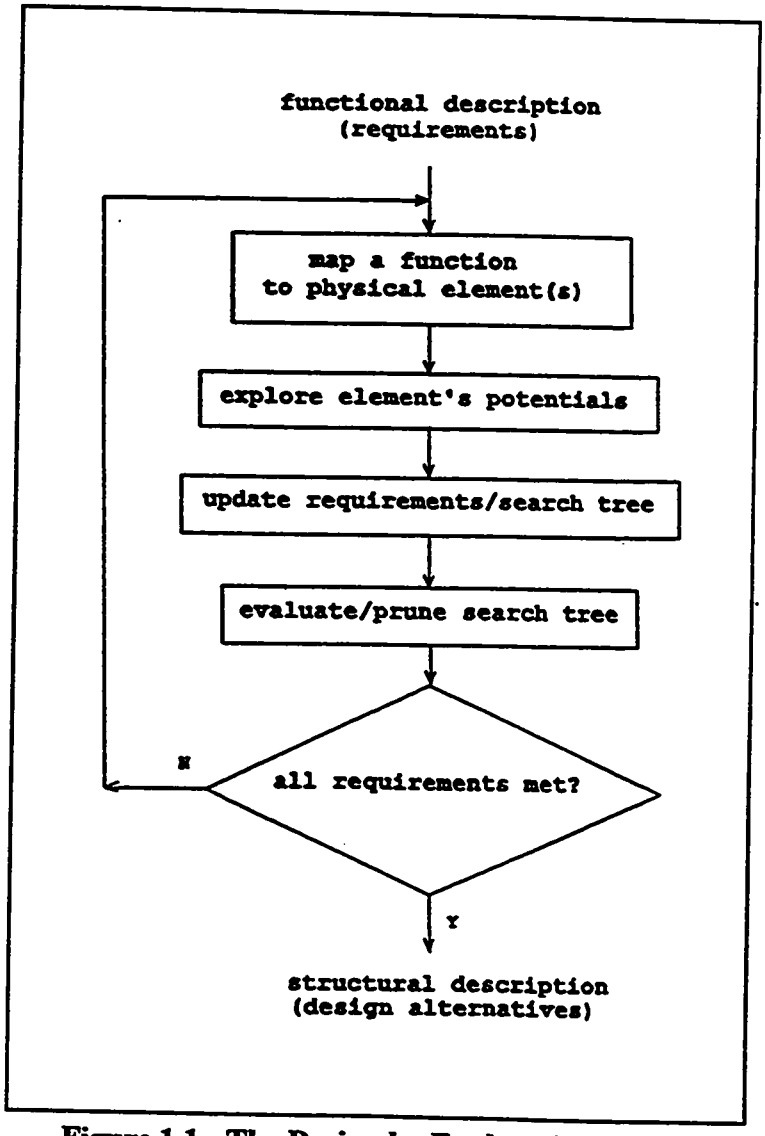


Figure 1.1 : The Design by Exploration Model (from [2])

description.

This research is based on the earliest works on conceptual design and its systematization, which was done by Europeans, especially Germans [3]. German researchers defined symbols to represent mechanical functions. The "Design by Exploration" model is schematically shown in Figure 1.1. This model uses the rule based design knowledge and generic algorithms to find feasible solutions for a design.

- "Quantitative Inference in a Mechanical Design 'Compiler' " [4]

This paper presents a computer program that takes a schematic of a mechanical system, as input, plus specifications and a utility function, and returns catalog numbers from predefined catalogs for the optimal selection of components implementing the design.

The paper introduces a theory for quantitative inference about sets of artifacts and operating conditions. The theory provides the basis for a mechanical design compiler which operates by eliminating unsatisfactory alternatives from catalog sets of artifacts.

A user of this compiler creates a schematic of the design by pointing in a list of elements. Each element normally includes a list of catalog numbers. Hence, each element represents the set of real artifacts purchasable by ordering from the catalog. Associated with each catalog number are specifications in the interval language. This language performs formal operations on real number intervals.

- "An Intelligent Knowledge Based Approach to Support Engineering Design Practice" [5]

The paper proposes an integration of Artificial Intelligence (AI) techniques with geometric

modeling systems which meet the geometric requirements of Computer-Aided Engineering (CAE) design practice. A Concept Model, which is a unifying model of different design models, is proposed in this paper to solve problems that arose from multiple models. In mechanical design, a number of different models are usually required (such as geometric models, kinematic models, dynamic models and strength models). The level and focus of the models are different. The Concept Model acts as a central design model, and four basic concepts of a design which are incorporated into it are a) design entities, b) relationships among entities, c) attributes of entities, and d) a set of reasoning mechanisms that are used in the design process.

In order to represent the design knowledge of an engineered product, the concept model provides two facilities: a) a library of basic parts or building foundations; and b) a concept language to allow the designer to express relationships among the basic building entities. Then, overall design is described as a structured hierarchy of parts and sub-parts, with a given set of rules that define the interactions among the various parts and sub-parts in the design product.

- "Design Synthetic Reasoning" [6]

In this research, a methodology, called design synthetic reasoning, has been developed to support computational processes for performing the task of design synthesis of mechanical machines. This method includes support for the following processes:

a) verifying that a design achieves the required behavior

b) deriving behavior characteristics of a design not specified as required behavior, and

c) justifying the function or purpose of components and relationships used in a design.

Design synthetic reasoning is based on an algebraic and predicate logic approach. In this

model, machines are modeled as systems decomposable hierarchically from both the views of structure and behavior. Systems and their decomposed structures are presented algebraically using set and list primitives while behaviors are represented using typed predicate logic. The function (the purpose) of subsystems in a machine system is also represented.

The process of design synthesis is modeled as a sequence of application of transformation rules. These rules either respecify, elaborate, reduce or reformulate the expression of required behavior, and ultimately allow matching and selection of structurally compatible machine components and relationships from a design library.

- "Development of an Expert System Shell for Engineering Design" [7]

In this paper, the development of an expert system shell which works for the preliminary phase (synthesis) of engineering design is described. In the development of this shell a morphological approach is used. Morphology is the science associated with the form and structure of a body or system. Using this approach, the results of a hierarchical planning is translated into a chart called a morphological matrix. This matrix is a table which creates visual representation of the various subsystems and their respective solutions in an effort to simplify the process of combining components to form design alternatives.

The shell has three basic components: knowledge base, context, and interface mechanism. The knowledge base includes the knowledge specific to the class of problems to be solved, and is organized into levels of abstraction. The context contains the information about the design problem currently being solved. The interface mechanism provides the design strategy.

1.4 Thesis Summary

This thesis is organized into eight chapters. In this introductory chapter the objectives of the CAID system are outlined. Then some related works are discussed.

Chapter 2 reviews the literature related to questions about design, creativity, design process, and the application of Artificial Intelligence (AI) and Expert Systems (ES) in the design process. In the second section of this chapter (2.2), different definitions of design, given by different design researchers, are compared, based on their methods. In Section 2.3, the definition and the process of design will be discussed by reviewing relevant literature. In a subsequent section on the design process, synthesis, analysis and optimization processes are cited. In the last section of this chapter (2.4), artificial intelligence and expert systems are described as strength tools.

Chapter 3 presents the terminology and characteristics of the object-oriented systems. In Section 3.2, the object-oriented approach is compared with the approach that uses procedural languages. In subsequent sections, four major concepts of object-oriented programming (i.e., abstract data types and classes, type hierarchies (subclasses), inheritance, and polymorphism) are analyzed. Then, using examples, it is shown how these concepts can facilitate the implementation of the CAID system. In the last section (3.9), the "Smalltalk" language is introduced, and the advantages of this language in implementing the CAID system are given.

Chapter 4 describes the philosophy of the CAID system and outlines its organization. In Section 4.2, different stages of the design process are discussed. The part design diagram (PDD) and the object relation diagram (ORD), which are used in the CAID system frequently, are defined in the subsequent subsections. Then, in Section 4.3, the organization of the CAID system is analyzed and

the system components are defined. These components are the Element Base, Design Knowledge Base, Connection Base, and the Design Engine. The Design Model is defined in Section 4.3.4, and its role is discussed. In Section 4.4.1, the Design Case is presented, and its relation with design model is defined. The users of the CAID system are introduced in Section 4.5. They are the Knowledge Engineer, the Senior Designer and the Junior Designer. At the end of the chapter (Section 4.6), the accessibility of the CAID system to the users is described.

Chapter 5 compares the theory of the conventional design with the object-oriented approach for the components of a gear train and shaft design. In Section 5.2, the conventional design theory is organized in a way that can be easily converted to the object-oriented approach. Three different theories in the gear design are considered for different type of gears. Then, in Section 5.2.5.2, a generalized method is introduced to analyze the general gear forces. In Section 5.3, shaft design is generalized to be applied to a shaft with any combination of mounted components on the shaft. The calculation of the partial length of a shaft for different types and orientations of the gear pairs are discussed. At the end of the section, the effect of the key on the shaft design is considered.

Chapter 6 discusses the implementation of the CAID system and its components. In this chapter, the method of identifying the objects and the classes is studied, and it is shown how the external behavior of the classes and the objects can be defined. Some criteria are suggested in this chapter to make a hierarchy of the classes. Then the subclasses of the Element class and Connection class are introduced and their hierarchies are made. The relationship of the ownership and inheritance is discussed in Section 6.4.1. The classes other than elements and connections are considered in Section 6.5. Then, using the object modeling technique, the structure and the behavior of the CAID components are studied.

Chapter 7 discusses how the CAID system works. In Section 7.2 the creation of a new system is explained. The method of storing and retrieving the design cases is discussed in Section 7.3. The protection of the CAID system is reviewed in Section 7.4. The design process of the gear train components is analyzed in Section 7.5. In Section 7.5.2 a general gear design process is explained; then the gear forces are analyzed in the subsequent section. Shaft minimum partial length calculation is mentioned in Section 7.5.4. Bearing design is studied in Section 7.5.5; then, shaft design is analyzed in Section 7.5.6, and shaft moments and shaft diameters calculations are considered as two subsections of this section. The keyway effect on the shaft design is the last section of this chapter.

Chapter 8, will conclude the thesis, and will give some suggestions for future studies.

Chapter 2

Literature Review

2.1 Introduction

What is design? Can creativity be taught? How do we begin the design process? Can engineering design be rationalized and systematized? These and so many other questions have been initiated in different papers. Erdman and Sandor in their book, "Mechanism Design Analysis and Synthesis" [8], go further: where does the computer fit into the design process? Can a design methodology or philosophy be formulated, practiced? Can scientific background and innate human intuition be augmented by a design discipline to enhance creative engineering performance. Answering these questions will help the researcher who wants to implement the design process on the computer.

Ulrich in his article "Intelligent Tools for Mechanical Design" [9] believes that, there are very few formal methods for solving design problems. In most cases this is because the problems are very difficult to completely and formally specify, and because the combinatorial complexity of design problems is severe. There are some kinds of problems on which formal methods have been applied, but they are exceptional and limited cases. However the development of design methods is possible. Further, there are thousands of different problem domains on which formal methods can be applied.

Major advances have been made in mechanical engineering design over the last few years. The progress is not only limited to advancing our knowledge of design, but also to clarifying the research methods necessary to study design. Finger and Dixon in "A Review of Research in

Mechanical Engineering Design" state that great progress is being made toward a better understanding of design, hence toward better design tools [10].

Due to the increase in computer processing power per cost ratio and improved technology, engineers are able to choose from a large number of different computer-based design systems and philosophies. Having noted that, however, success is not a matter only of how hard you work, but also how smart you work. There is no productivity improvement in mechanical engineering designs without getting help from computers. Ulrich in the same paper states that; computer-aided design systems help with drafting and some kinds of analysis, but little else. To improve computerized design methods, advanced interfaces are necessary so that designers can tell computers what they want. Smarter synthesis programs are needed so that the computers can do what they are told to do. Better simulation methods are required so that the computers can test what they have done [9].

About the cognitive processes, Dixon in his paper "New Goals for Engineering Education" [11] notes that very little is known about the kinds of processes used in mechanical design. Experienced designers can create better designs because their knowledge is broader, more general, and more abstract. Most individual designers do not explore alternatives well or thoroughly, but instead try to fix or patch up their initial ideas.

The complex process of creative engineering design is subject to infinite variations. One objective in this study is to present a general guideline, in a form which is readily kept in mind by a designer, and can thus serve as an aid of broad applicability in practice [26].

Some research publications discuss research methodologies used to approach design problems. Daizhong and his colleague in "Development of an Expert System Approach to an Engineering Design Procedure" [12] describe that, in a mature field , the researchers will share a

common view of what appropriate research methodologies are, what the difficult, unanswered questions are, and what constitutes high-quality research. In the emerging field of design research, no such consensus exists. This makes the design field more exciting and revolutionary with lots of new ideas.

This chapter will discuss the definition and process of design by reviewing relevant literature. In the next section different definitions of design, given by different design researchers, are compared based on their methods. In a subsequent section on the design process, synthesis, analysis and optimization processes are cited. In the last section of this chapter artificial intelligence and expert systems are introduced as new power tools.

2.2 What is Design?

Design activity is a distinguishing characteristic of engineers as compared to scientists. Engineering design is a process used to create objects or systems that perform a desired task. However, most scientific aims are to discover why something, in a real world, behaves the way it does. The facts discovered by scientists originate outside the human mind. In contrast, the origin of an engineering design comes from within the human mind.

Soldan in "Engineering Design: What is it?" [13] initiates the following question : "Is engineering design something that can be learned in school?". His answer is definitely yes. However, good designers are not born rather are made by lots of experiences and failures. Creativity is a crucial factor in creation of novel designs. Engineers who enter management or non-engineering careers have been known to call upon their design-oriented manner of thinking to assist them in their new

fields.

In the same paper Soldan defines the engineering design as the process of devising a system, component, or process to meet desired needs. It is a decision-making process (often iterative), in which the basic sciences, mathematics, and engineering sciences are applied to convert resources optimally to meet a stated objective. The establishment of objectives, criteria, synthesis, analysis, construction, testing, and evaluation are the fundamental elements of the design process. The accuracy of the design decisions based on the model can not be more than the accuracy of the model.

Maher and Longinos in the "Development of an Expert System Shell for Engineering Design" [7] give another definition. They argue that design is a process of producing a description of a system or process to satisfy a set of requirements. Both synthesis and evaluation methods are required for identification and composition of components of design solutions. Such methods can provide a systematic approach to design. Alternative solutions can be evaluated based on a discourse of criteria and value. Without sacrificing the benefit of a systematic approach, the designer can guide the method even with qualitative knowledge using knowledge based techniques.

They assume design to be a process during which design descriptions are generated to satisfy design intentions, where identifying the design intentions is as important as identifying the appropriate design description. As the design process continues (through several levels of abstraction) more information about the requirements as well as the evolving design description is available. In the early stages of design the design knowledge is mostly qualitative, and the major components and subsystems of the design are identified in these stages .

In another paper [14], they describe design as a combination of art and science which is

perhaps one of the most important and difficult tasks an engineer performs. Recent researches on this creative process result in developing a number of design methodologies for different engineering fields. There are few successful applications of the above design methodologies in the literature, although substantial efforts have been made to produce a structured approach to the engineering design process.

The design process can be considered as a search process, where the search method operates in the design space to locate the solution. A design goal is a main concern for the design process. A design goal can be decomposed into a set of subgoals, whether it is functional or physical. The design space for a class of engineering systems can be built by a set of the design subgoals, their solutions and the relationships between them. For example, in designing a gear train for a transmission system, design goals include: design some shafts, gears and select proper materials, design a housing, etc. Knowledge of these subgoals and the possible solutions provides a basis for reasoning about design synthesis.

2.3 Design Process

How do humans create designs? This question is the main concern of many researchers from different fields. They have studied the processes, strategies, and problem solving methods that designers use. Artificial intelligence has an important role in this kind of research. Recent research has focused on categorizing, studying, or modeling the cognitive processes, in contrast to the earlier work on creativity techniques. A cognitive model is a model that describes or simulates the mental processes used by a designer while creating a design.

By recording the actions of the designer, it has been proven that much of the design process is a mental process [10]. Literature review on design research shows that the preliminary design stage has been studied in the most cases, while designers might be using different design strategies during each separate design stage. To be able to record the mental activity of the designer, usually the designer is encouraged to think aloud and is questioned when information seems to be missing or incomplete. The requirement to verbalize nonverbal designer activities may interfere with the design process itself.

Although there is no single design process or design strategy, there are some well-defined questions to study the designer mental activity. For example, Finger et al [10] discussed differences between the strategies used by expert designers performing a familiar design task with their strategies performing an unfamiliar design task.

In the same paper the authors explain three groups ideas on controlling the design process. One group believes that the design process should be chaotic and creative, another group believes that it should be organized and disciplined, while the third believes that no process should be imposed on a designer. However, better computer-aided design tools are required in mechanical engineering, and to create better tools requires knowledge of how designers design. Having this knowledge make it possible to achieve a systematic methodology.

As Erdman and Sandor [8] mentioned in their book, creative design is not a one-way, single-pass effort. It is often necessary to retrace one's steps: feedback and iterations may occur at any stage. If resynthesizing cannot correct undesirable responses of the analysis stage, a new concept should be considered. The problem should be redefined, if a suitable concept can not be generated. This cycling can continue to create superior designs.

2.3.1 Cognitive Modeling vs. Computer-based Modeling

Mechanical engineering is a very old discipline comparing to cognitive modeling. A better understanding of designer strategies is being achieved due to growing research on cognitive design processes. The better understanding of designer strategies, the better computer-based tools for designers.

The main goal of cognitive research is to simulate computer-based models that describe the human skills for problem solving. The models describe the designer skill processes which have been discovered by cognitive research. It is capable of analyzing and predicting human design skill since the model simulates designer functional mechanisms. As mentioned this research field is relatively new, hence, there are only a few papers available about this recent research topic.

Finger and her colleagues [10] compared the cognitive model with a computer-based model. They define the cognitive model, whether given in words or in computer code, as one that attempts to describe, replicate, or simulate human mental processes. But the computer-based model, in contrast, as one that expresses a method by which a computer may accomplish a specified task. Human thought about the design process may partially be simulated in a computer-based model, but with questionable value. Some designers have attempted a successful methodology which is derived from a computer-based model. However, they (Finger et al) conclude that cognitive models are concerned with how humans do design whereas computer-based models are concerned with how computers can design or assist in designing.

2.3.2 Conceptual Design & Synthesis

Conceptual design is defined as the transformation from functional or behavioral requirements to structural descriptions. For example, if the desired function is to transfer a rotation from one axis to another one (and possibly change the speed of rotation), then possible embodiments are some shafts, gears, etc. Such a configuration design is defined as the transformation from a physical concept into a configuration with a defined set of attributes, but with no particular values assigned. For example, a gear train is a physical concept with the possible configuration of some shafts and gears.

Configuration design can be an assembly of standard components (e.g., gears, shafts, bearing, and motors), or redesign of a non-standard component (e.g., a bracket). Computer-based models for conceptual design are in a more embryonic state than those for configuration design [10].

Decomposing the system into subsystems and components is referred in some literature as synthesis [27,28], and can be viewed as a search for solutions throughout the constrained design space. Optimal solutions can be selected by evaluating different solutions according to some optimization objectives. For given requirements, a single model consisting of both synthesis and evaluation can produce alternative designs. The experienced designer can develop the knowledge based synthesis program using this single model as an environment. To produce design solutions, the novel designer uses this knowledge base.

Although synthesis is a solution search through design space, this space is not identified explicitly, rather it is implicitly developed during the design process. However, the relevant design space representation exists explicitly in the design program. Using the knowledge based approach

to design makes the solution searches more efficient through the knowledge based design space. This knowledge is developed and expanded as the designer experiences more designs [14].

Decomposition of the design problem into independent subsystems is the most popular synthesis method, but it is not the best one for all kind of designs. In some cases building up the system by starting from details is a better approach [7]. Synthesis process for design in our approach will be discussed in detail in chapter 3.

2.3.3 Analysis & Optimization

One should note that analysis supports design, and not the reverse. In computer-based design the process of making design decisions is different than analyzing the design. Design evaluations and decisions may be based on information which is provided by analysis. This analysis of design performance is a procedure along one or several dimensions of the design.

To be sure about the accuracy of design performance, the analysis procedure should be applied on the solution of any detailed design stages. The design can be guided and the redesign decisions can be made by evaluating the performance information which is provided by analysis procedures. In engineering, evaluating trial designs (i.e. applying analysis procedures) is one of the important tasks.

Although there are many analysis procedures to predict the performance of the design along different dimensions, to make them more user-friendly for designers, a good interface is a must for these analysis procedures. However, by increasing critical decisions in recent design methodologies, better analytical tools are necessary. It will be possible to make these critical decisions based on

qualitative information which is provided by good analysis procedure. The modern analysis procedures are helpful in exploring all alternative design solutions efficiently. Analysis procedures should apply to all the stages of design, but most of these procedures are built for the detailed design stage. For example there is no procedure, so far, to evaluate conceptual design.

Optimization is a tool that enables a designer to locate the best alternative solution. When it is appropriate, these closed-form procedures are used in the design models. There are three conditions for using optimization methods: a) having a single objective function which depends on design variables; b) design constraints must be known; and c) all the constraints must be quantitative. Most of the optimization methods try to maximize or minimize the objective function.

There are not many optimization methods which can apply to realistic mechanical design problems. But there is much research recently going on to extend new methods to be useful for the mechanical engineering designer. Designers should know when, which, and how to apply these powerful optimization methods. There are some knowledge-based approaches for parametric design which use some other methods than regular optimization. Dominic [15,16] reports one of these approaches that has an interface to its own formulations and methods for finding optimal or suboptimal solutions.

Finger et al [10] believe that a computer-based model for design is generally specific to a well-defined class of design problems (i.e., parametric, configuration, and conceptual or preliminary type design). At the beginning of the design process, in parametric design, the general shape and attributes of the desired part or component are known. Parametric design is not as complicated as other classes of design problems; rather it is a method to choose appropriate value for the parametric design variables. It is important to know that all of these variables are not numeric (e.g., a shaft or gear type

or material).

There are many types of tools accessible to designers. The background of the designer as well as the type of design, and the design environment defines how useful a particular tool is. Storing the solutions of the designs which are already completed and retrieving them as starting points for new similar design is the most useful tool in the design process. Having the design in a computer database can also allow several engineers to share the data and can speed the design and manufacturing process. If the designer is familiar with part design, but not good enough with structure or configuration design, he needs tools that will help to retrieve and assemble the elements in a simulation. Working in a new environment and domain needs tools with user friendly interfaces and more control on the constraints.

Most current design methods do not allowedfor the evaluation of designs that do not have instances. But new research on the computer-based models is going to make explicit evaluation for these type of designs possible. As Finger [10] mentioned, better evaluation of incomplete designs will not only make computer-based design more practical, but will also enable designers to explore more configuration alternatives. That is because searching the design space is not an easy task, and it is almost impossible to search the entire space within limited time and cost. Hence, directing the designer through the design space toward feasible and optimum solutions will give the opportunity of considering more regions in the design space. In another words, instead of searching the whole design space through all unfeasible solutions, the designer will be guided directly to the feasible solutions.

2.4 Application of Artificial Intelligence and Expert Systems in Design

Representation of designs graphically using computers is another major achievement in addition to that of programming analytical procedures for designs. The advanced and inexpensive technology of current computers make computer aided design tools more functional than traditional tools such as paper and pencil. Further, this new technology makes better computer aided geometrical models, greater semantic integrity, and superior abstraction hierarchies. Eastman [17] is among the first researchers in design who initiated the idea of integrated product model, as opposed to a Computer Aided-Design (CAD) database, for mechanical designs. Since then, researchers progressed a great deal in creating integrated models. These models, which are called engineering databases or product models, consist of all kind of design models such as geometry representations and semantic knowledge [10].

Yoshikawa [18] in a paper entitled "Automation of Thinking in Design" has looked at the results of design research by asking the question: what in design do we understand well enough to be able to automate? Whether one agrees that the goal of design research is to automate design, answering the question makes clear those aspects of design which are well understood and those which are not. The criterion which is defined in this paper, for measuring the success of design is a good tool for evaluating the design, but there is no explicit methodology.

In most engineering fields like mechanical engineering, CAD systems are widely used by designers. Purely geometric information is not adequate, although the main goal in developing CAD systems is to ease the geometry handling. In a design case, when only the physical aspect of the design is considered, this kind of information will be enough.

The complexity and the cost of engineering designs are increasing and designers are dealing with more complex constraints. Increasing economic competition demands faster, more efficient and productive design and manufacturing processes. Lead times and product costs are decreasing due to the application of computer tools in the design-analysis-manufacture stages. These improvements are better in the engineering design process than other fields. Documentation of design information and the design representation are among the most important and helpful tools [5].

Advances in computing technology have considerable impact upon Computer-Aided Engineering. The desire for more extensive computer-aided support of the engineering design process has led to attempts at developing and extending more intelligent systems. The most important ones are expert systems which are computer programs with a specific and available knowledge of a human expert. This specific knowledge can be used together with a set of rules based upon the expert's learning and experience. Most expert systems are outside the engineering field such as medical diagnosis. These systems become more powerful when they use the knowledge of several human experts.

The next generation of these expert systems are the learning expert systems. Such systems have the potential to obtain the formal and available knowledge and experience of the staff of a design office. Without this ability, at most they have only the extracted knowledge which they got in the beginning [19].

No matter how complex the problems are, the concept of design problems and their design method should be well defined if the aim is to use expert systems to solve these problems. There are many engineering design problems of this kind, but still expertise is required to achieve the best design [12]. Almost all of these types of problems are outside the mechanical engineering design

field, e.g., circuit design problems.

It has taken several years of experience in the engineering design field to learn the preliminary phase of this process. Implementing such a learned technique in an expert system can be a powerful tool for the engineering design process. Using a hierarchical approach following the same procedures humans do makes expert systems more powerful.

Artificial intelligence or thinking machine is another idea which simulates a sort of human brain activity. It will take a great deal of research effort for AI researchers to prove how successful this idea will be. Any attempt to implement even the simplest task of the human brain capability on computers induce programmers to appreciate the power of the human brain, compared to that of the present day computer [19].

Researchers in AI are trying to understand and formulate human intelligence, so that it can be implemented on computers to facilitate thinking in the relevant domain of application [5]. To improve the ill-structured phases of the engineering design process implemented on computers, artificial intelligence and expert systems are very powerful tools [7]. This improvement can be done, for example, by using knowledge base expert systems, and rule-base reasoning or heuristic search methods of artificial intelligence.

In some fields, such as circuit design, the design component attributes (i.e. physical, functional, and logical) can be more easily represented formally. That is because the parameters that apply to electrical components are quite limited, as are the inputs and outputs that relate them. However, in mechanical engineering design, the intensive research efforts in this regard have resulted only in computer-based models for the geometry of the designs. So there is no formal presentation for the physical and functional attributes of mechanical designs, except in limited domains such as

kinematic linkage design.

Recently, Knowledge Revolution has released an application software, namely, Working Model. This software combines motion simulation technology with dynamic simulation. It uses kinematic and Newtonian mechanics laws and provides an animated graphical interface. Working Model simply works by defining rigid bodies and constraints; then it calculates the motion of interacting bodies using numerical analysis techniques. Working Model is the first software program of the kind in the market.

The environment of the design process is very important, especially when the design system is computer-based. Present tools for design are not standard, so transformation of the design from one representation to another is a time consuming task for a designer. Common representation among different tools may solve only the transformation problems. However, more research work is necessary for the coordination of the tools with the designer. Finger [10] mentioned that design is a search within a constraint space, and the environment enables the designer to navigate through the constraints of the design space.

Chapter 3

Object Oriented Systems

3.1 Introduction

The idea that object-oriented problem solving represents a methodology that is quite different from the approaches supported by structured programming languages is accepted by many related field researchers. Wiener in his paper "An Introduction to Object-Oriented Programming and C++" mentioned that the powerful features of object-oriented languages support concepts that make computer problem solving a more human-like activity.[29]

Object-oriented programming is a powerful method for structuring software systems. It encapsulates the data and the operations in objects with a cleaner interface. Based on the special characteristic and functionality of these objects, a hierarchy of the object classes is built up. Large systems can easily be constructed using reusable components in this method, and these systems can be changed according to new requirements. [30]

Until recently, object-oriented programming has only been used by experienced programmers who understand object-oriented concepts due to the experience of programming actual systems. Very little formal theory about these concepts has been developed. To have a solid foundation for refinement and further development of object-oriented programming it is necessary to have a more formal understanding of its basic concepts.[31]

The CAID system has been implemented using the object-oriented approach. To elaborate on the use of object-oriented programming concepts in CAID, the design of a gear train is used as a medium.

3.2 Object-Oriented vs. Procedural Languages

A software program developed using procedural languages is coded as a collection of functions and data. The problem which is supposed to be solved, is divided into several tasks, each of which is broken down into smaller tasks. This process continues until each task can be represented by a single language statement. This approach asks "how" before "what". When designing a gear train using this approach, a programmer should ask 'how the stresses of a shaft will be analyzed?' before asking 'what is a shaft's role in a gear train?'. After knowing "how", the programmer starts implementing the functions, and then specifying the type of data.

Solving a gear train design problem using procedural languages, a programmer has to break down the design problem into smaller ones, like shafts design, gears design etc. Although there are similarities in the design of different type of gears like spur, bevel etc, a separate procedure is required to design each type. The program continually increases in size as a programmer adds more code for the design of additional mechanical components in order to create an integrated mechanical design system.

By contrast, in object-oriented programming, to know "what" is more important than to know "how". This approach starts at a more abstract level. The first question to be answered should deal with the goals of the application. The programmer defines the objects and their relationships. In the application, each object should know how to perform its own operations and store its own information. Hence, the programmer should determine the operations to perform and the information that results. The responsibility for those operations and the information should be divided among the objects in the application. [32]

Object-oriented programming centers around four major concepts: encapsulation of objects, class hierarchy, inheritance, and polymorphism. These basic concepts are introduced in the following

sections.

In object-oriented programming abstract data types are basic elements used in the development of systems. An abstract data type refers to a model that consists of the type of data and an associated set of operations. The behavior of the underlying type is characterized by these operations. A class definition is an abstract data type in most object-oriented languages. This definition describes the behavior and the interface to all the operations of the underlying abstract data type. It also specifies the implementation details or data structure of the type, which are accessible only within the scope of the class [29].

Two relatively early motivations for object-oriented programming fundamentals are:

a) the difficulty of designing and maintaining large software systems, and

b) the necessity of structuring the information related to a specific subject in a single entity.

A programmer aims at facilitating a design, maintaining large software systems and enclosing information and properties associated with a given concept in a single entity. In this thesis, it will be shown that these goals are achievable with new concepts like encapsulation, hierarchy and inheritance in object-oriented programs. [33,34]

3.3 Objects

A system, in object-oriented programming, is a collection of objects (Figure 3.1). Each object contains some data which is stored in variables, and some procedures which acts upon this data. In this thesis the Smalltalk terminology will be used, hence the procedures of the object are called methods. The content of the object variables can be a basic data type or a reference to another object. In a pure object-oriented language, such as Smalltalk, all the data are represented by objects [35], consequently the first case does not exist.

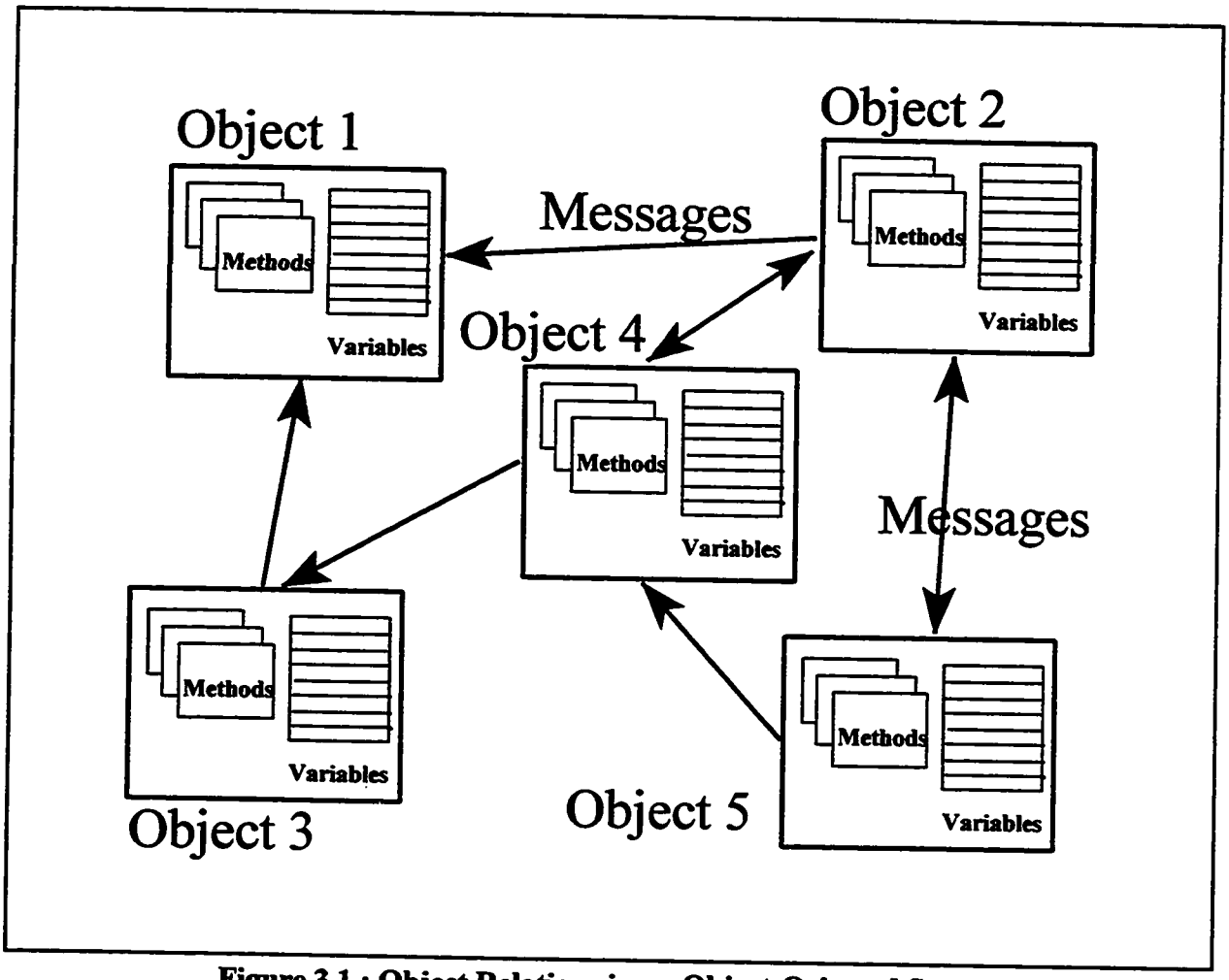


Figure 3.1 : Object Relations in an Object-Oriented System

Objects, in object-oriented programming languages, may be complicated identities. They can change their states, yet maintain their identity. Hence, it is essential to be able to deal with the dynamic structure of references ("pointers") between objects. The object behavior that can be observed from another external object should be distinguished from its internal structure.

Each object has its own unique identity. A unique name will be given to each object to differentiate between them. In this thesis, a simple rule is made to name objects. The type of the object with a capital letter at the end represent an object's name (e.g. gearA or shaftC). The name will start with a lowercase letter, which is because of Smalltalk regulations. For other entities, other

than objects (i.e., methods, variables, etc), meaningful names are more used. If the name consist of more than one word, they will be connected. For easy reading, the connected words will be capitalized except the initial word (e.g., getDiameter).

When a gear is to be represented in an object-oriented system, all related information should be implemented in the object. Different types of variable (geometric, stress and internal) store and represent its states in all aspects. The geometry of the gear should be represented by some variables such as pitch diameter, face width, type of teeth, number of teeth, etc. By knowing the values of these variables it should be possible to draw the gear. From the strength point of view, there should be some other variables (like gear material, allowable stress, induced stress, applied torque, etc) to give the designer the ability to analyze the gear design. There will also be some other variables that point to other elements connecting to the gear (e.g., supporting shaft and meshing gears). Having access to these connected elements makes the analysis of the transmitted forces and torque possible. Other than these variables which represent the state of the gear, there will be some operations that are called methods, which change the state of the gear. There is a relation among the force applied on the teeth of the gear, pitch diameter, torque and its pressure angle. Hence a method can be implemented to calculate the force applied on the teeth according to the values of other related variables. The gear is a complete entity when including these variables and methods, which are necessary to represent the state and behavior of the gear.

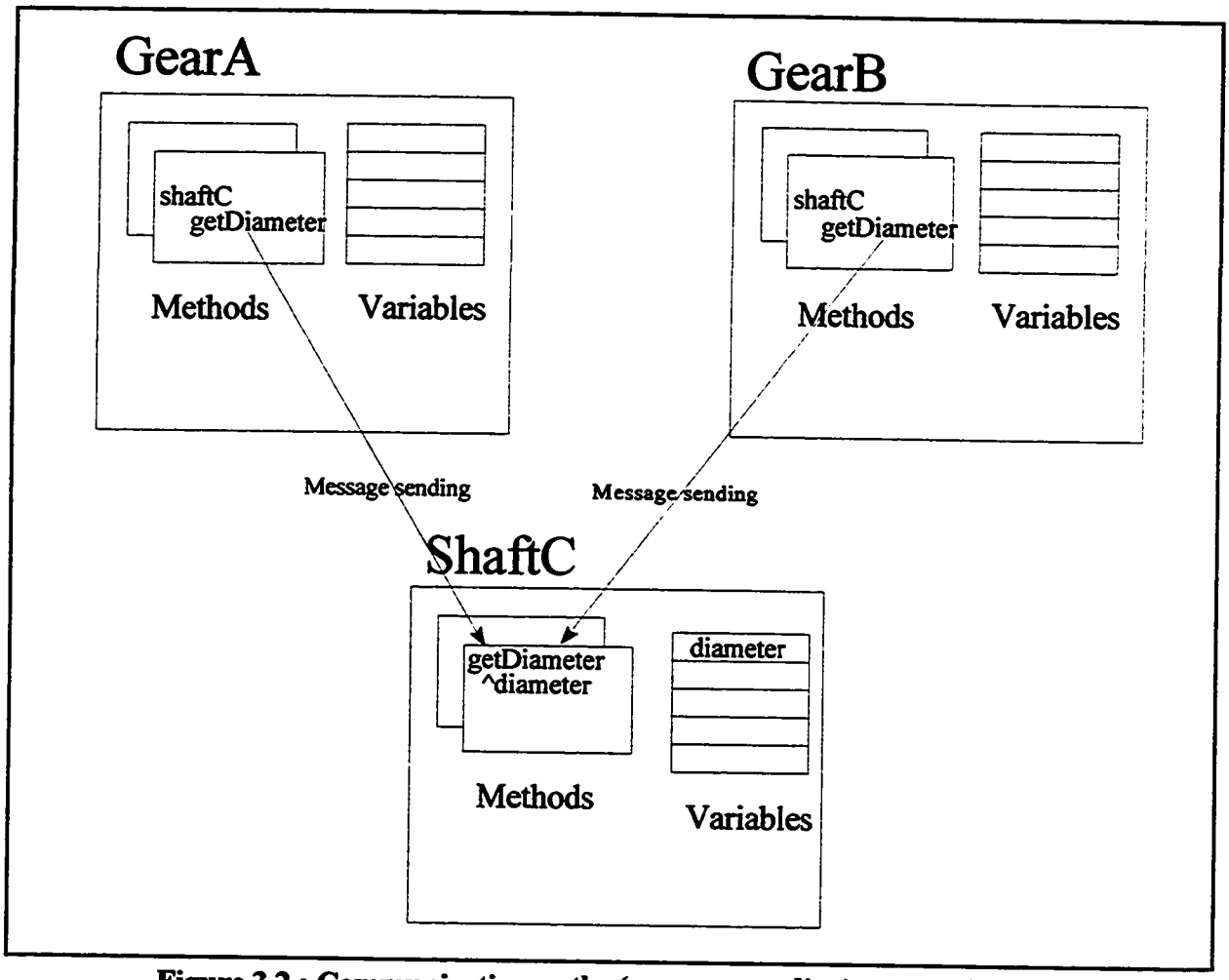


Figure 3.2 : Communication paths (message sending) among Objects

In Object-oriented programming, the data of one object is isolated from other objects. That means, the variables of one object cannot be accessed directly by other objects. The only communication among objects occurs by sending messages to each other [36]. Object methods (procedures) can be executed by sending a message to call them. For example GearA and GearB that are mounted on the same shaft (Figure 3.2) send the message 'getDiameter' to ShaftC requesting the method 'getDiameter' to be executed. In the 'getDiameter' method, the variable 'diameter' of the shaft is accessible. By returning the value of this variable to the sender of the message, GearA or

GearB, the gear can have indirect access to the diameter of the shaft. How the shaft obtains and returns its diameter is of no importance to the gear, it may be a complicated method that calculates the diameter using a Finite-Element Method (FEM) approach, or it may simply return the value of the variable 'diameter'. The only information which is of value to the gear is its supported shaft diameter. This is the only means that one object can have access to the variables of another object. The set of object methods is a good interface with other objects. This is a powerful protection mechanism through which all access to the inside of an object is carried out through its well defined methods. This allows the object behavior to be separate from its implementation (the variables and the methods).

3.4 Encapsulation

Conceptually, the word encapsulation means to put a capsule around data and procedures of an object. All knowledge and operations which are considered as contents of an object, are encapsulated in the body of that object. Dealing with the items in the capsule as a single unit is easier than dealing with them as disjointed items. Encapsulation puts many ideas and operations into a single unit which is called an object. It simplifies the complexity of the system. [37]

The system can be simplified by encapsulating the related items. The capsule should be more than a concept for other objects. As a principle of information-hiding, each object has two quite distinctive aspects, a public interface and a private representation. [38] The only mechanism to access the objects through their public interfaces is called the message-send. [39] The object to which a message is sent is called the receiver. The purpose of sending messages to other objects is asking for a procedure execution or the receiver's private data. From an internal point of view, the

receiver executes a method which is an algorithm for manipulating the data. If there is no result to return, the default result will be the receiver itself. For example, in Figure 3.3 each of GearA and ShaftC are encapsulated objects. Each can access the other's encapsulated information only through sending messages to that object. The sender can pass some data to the receiver as arguments and receive some results from the receiver.

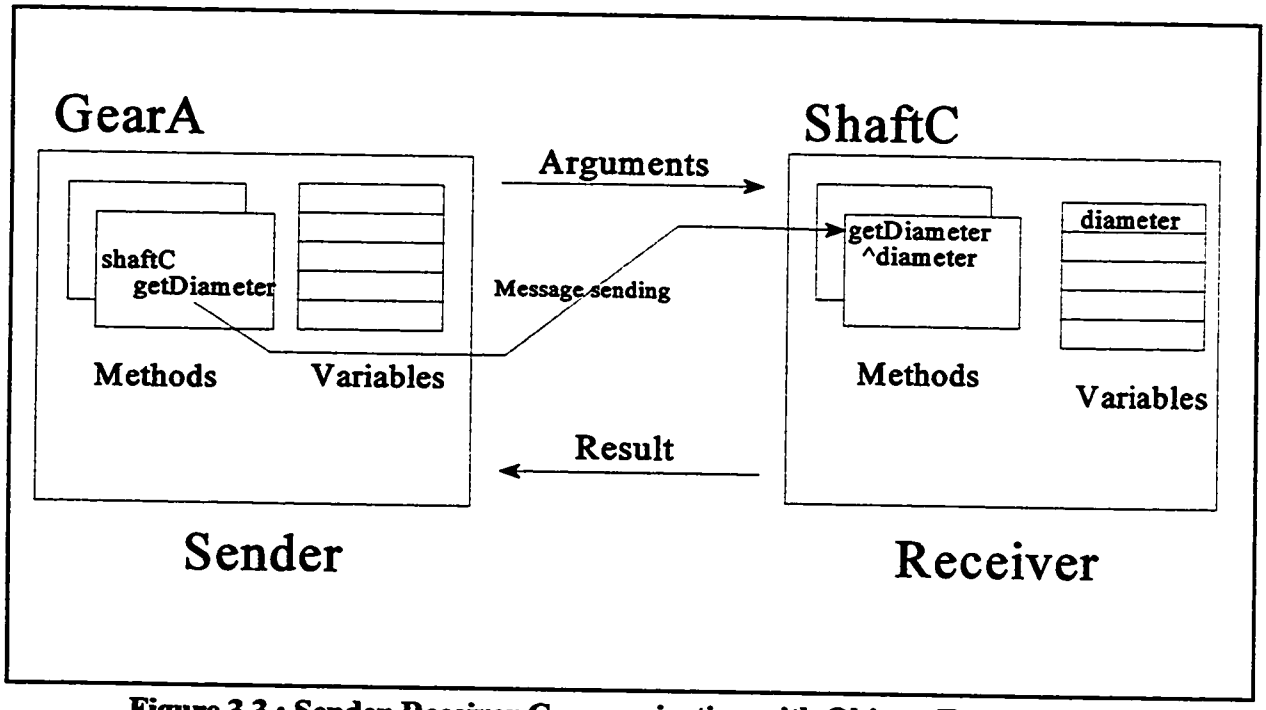


Figure 3.3 : Sender-Receiver Communication with Objects Encapsulation

Information-hiding distinguishes between what tasks an object can perform and how it performs them. Such a separation allows a programmer to design more maintainable and extensible applications.

Encapsulation helps the programmer to take the abstract view of the application, and leave the details until later stages such as the object design stage. Hence, in the first stage, the concentration should be on the layout of the application.

Encapsulation has another advantage in the stage of the object design. The design of one

object does not depend on the design of other objects, hence each object can be designed independently. If a better algorithm, or method, is found, or new hardware installed, changing the internal structure of an object does not affect other objects. Changing an object's internal structure will not affect other objects as long as the interface remains unchanged.

Encapsulation of objects helps the programmer to design the objects in two independent stages:

1. Determine related information and procedures, and encapsulate them in an object.
2. Determine the information and procedures that other objects require from the object, and hide the rest.

3.5 Classes

Although objects are completely distinct entities, many of them are quite similar to each other. In a mechanical engineering design, the designer often needs elements such as shafts, gears and other mechanical components. Shafts do not behave like gears, but different shafts do behave like each other.

Grouping objects into classes is useful to describe all the objects in a system. The objects are the instances of the classes, and instances of one class have the same methods, variables and body. For example, all the spur gears will be the instance of the SpurGear class, and have the same methods and variables of their class (i.e., SpurGear class). Although each object has its own variables, the names and types of the variables are analogous among all the instances of a class. In object-oriented languages, a class contains a number of variables and method declarations (Figure 3.4), and a system consists mainly of class definitions. Hence the exact information necessary to

create a new object can be found in its class definition. Instantiating an object from a class is carried out by using the method 'new'.

Instances of one class share the same behavior, so any number of similar objects should have a generic class. As an example, SpurGear class represents the generic form of all spur gears. Classes are like templates in assembly lines. They can be used to produce as many objects as necessary.

3.6 Hierarchy

Abstract data structure techniques are also used in object-oriented programming. A concrete internal representation in the class implementation of objects provides this abstraction. Arranging the classes in a hierarchy provides a great power in object-oriented programming languages. This class hierarchy is the same as living organisms arrangement in biology. Classes close to the top of the hierarchy are more abstract, and classes down the hierarchy far away from the root (top) are more specialized. For example, animal and plant are more abstract than fish and tree. In mechanical engineering design applications, elements like gears and shafts can be classified and arranged in the same way. For example, in Figure 3.4, different type of gears and bearings are classified under their generic superclass, which are Gear and Bearing.

A class with a higher level of abstraction compared to another class is called a superclass of that class. On the other hand, the class which is more specialized will be called a subclass of the higher abstracted class. For example in Figure 3.4, Gear class is a superclass of all types of gears (i.e., BevelGear, SpurGear, etc), and those classes of gears are subclasses of Gear class. In Smalltalk each class has only one immediate superclass with class Object at the top of the hierarchy. Also each class can have many subclasses, or none.

There are classes in the class hierarchy that do not have any instance, like Gear class in Figure 3.4. These classes are called abstract classes. The abstract classes are useful for developing generic problem solutions. A programmer can implement methods that represent the part of the design solution which is similar for all type of gears in Gear abstract class. Later, by adding other methods which represent specific solutions for each subclasses, like SpurGear class, the design solution will be completed.

3.7 Inheritance

Inheritance is another abstraction mechanism which is supported by all object-oriented programming languages. Inheritance allows a programmer to define a new class as a refinement of an existing one. The new class captures the similar characteristics and specifies only the differences. Repeating this new class definition results in a complete inheritance hierarchy. It is important that the two ends of this relationship be distinguished. They are already defined as superclasses and subclasses. Reusing the code which already exists in superclasses is the main purpose of inheritance. By adding new behavior, a subclass defines its own unique kind of object. Each class, in Smalltalk, can have only one superclass. Whenever a subclass is defined, all instance variables and methods of its superclass are available for its use. This allows rapid designing of subclasses.

The main idea of inheritance is that the subclass inherits the variables and the methods of the superclass, and adds some more in order to become the required class. For example, in Figure 3.4, SpurGear class inherits the variables (e.g. noTeeth, diameter, width, etc) and methods (e.g. 'A1','getDiameter', etc) of its superclass Gear.

The amount of code in a software system can diminish considerably by using this powerful mechanism. As mentioned, an object is instantiated from its class using method 'new'. Then it is initialized by another method usually called 'initialize'. Suppose, an 'initialize' method is to be

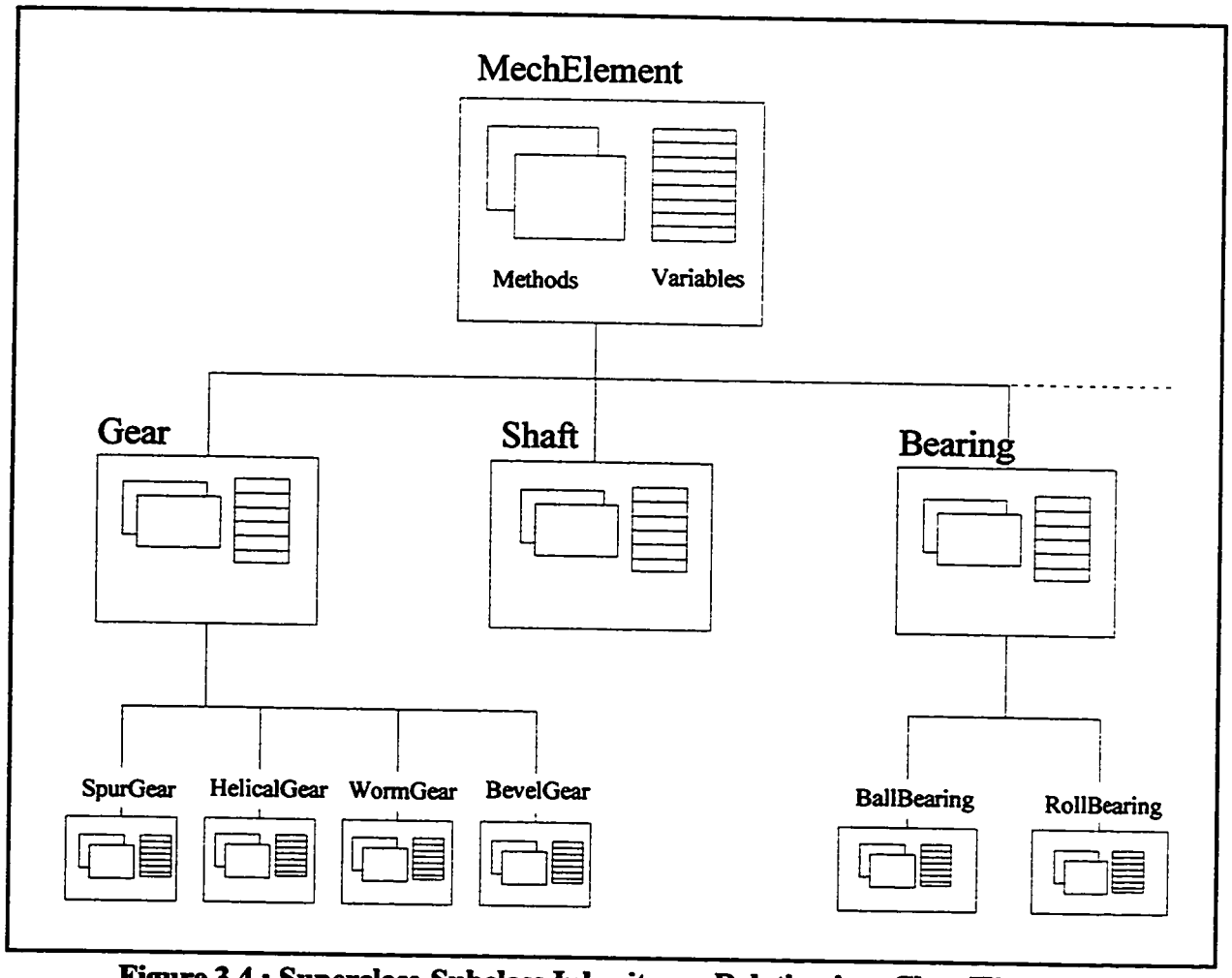


Figure 3.4 : Superclass-Subclass Inheritance Relation in a Class Hierarchy

implemented for SpurGear class. A spur gear could be initialized in the following ways:

a) Having the same steps as initializing a gear in general which is already implemented in Gear class, the superclass for SpurGear, there would be no need to implement the 'initialize' method in SpurGear class. It will be inherited from Gear class.

b) If there is no 'initialize' method for a gear in general, and each gear has its own initialize method, then an initialize method for SpurGear class has to be implemented.

c) When the spur gear initializing method is different regarding the gear initialization in general, the initialization method will be overridden. This is when there is no similarity between the initializations.

When initialization of a spur gear is different than general gear initialization (as in case c), but involves some similar codes, then, to avoid the repetition of the codes which already exist in the general initialize method in Gear class, a new initialize method can be implemented in SpurGear class. In this method, the superclass (Gear class) initialize method is called, then other codes that complete the initialization of spur gear will be implemented.

3.8 Polymorphism

Different responses to the same message by instances of different classes is a mechanism called "polymorphism". This abstraction is made possible by distinguishing between messages and methods. The object which sends the message should know that the object which receives it can respond to the message. However, the message sender does not need to know to whom the message is going or the mechanism by which the receiver responds to the message.

Different objects respond to the same message with their own unique behavior. Hence, new classes can be integrated in the application as long as they implement the methods necessary to respond to the messages required by the application. In this way, name overloading will be avoided, and the system will easily be extended.

Polymorphism allows the programmer to use the similarities between different classes. For

example, the weight calculation of all mechanical elements depend on their volumes and densities. An appropriate method for one element might not be good enough for another because the volume of each element depends on its geometry. Hence, each element can have its own method to calculate its volume and weight, but with polymorphism all of these methods could have the same name calculateWeight. By sending this unique message calculateWeight to each of those elements, their weight will be calculated according to their volumes and densities.

3.9 Smalltalk

Smalltalk is referred to as a pure object-oriented development environment. The word "environment" means that a hierarchy of classes in Smalltalk is already created. There is no such independent program for a compiler to compile. However the programmer can develop his own classes and codes on this hierarchy of classes, and any other programmer can use these new classes. The class library of Smalltalk is rich, it is easily reusable and extensible. It relieves programmers from involvement in low-level detailed programs. Almost any type of application can be developed by this library. This is a tremendous advantage in programming development. [40]

As a pure object-oriented environment, systems developed using Smalltalk are forced to adhere to the four concepts of object-oriented programing (i.e., abstraction of classes, hierarchies, inheritance, and polymorphism). In Smalltalk everything is defined as an object. Even a class is defined as an object. Integers are objects that have methods that define how they react to a message like "add", "subtract", etc. Object variables are its private memories, and in Smalltalk they are called instance variables.

Unlike procedure calls in conventional programming languages, the messages specify their

associated operation by a message name which is called a selector. The receiver class interprets the message selector. Other than the message name, i.e., selector, each message has some arguments, if required.

Methods as step-by-step algorithms, perform the requested operations in the manner specified by their internal representation. They are not part of the public interface of an object, rather they are always part of its private representation. A method name matches the selector. There is no need for all behavioral aspects of an object to be presented by publicly accessible methods. For implementing publicly accessible operations, each object can send private messages to itself. For example, in the "calculateWeight" method, the programmer could send a "volume" private method to itself to calculate its own volume according to its geometry, then multiplying this volume by its density to obtain its weight.

In Object-oriented languages (e.g., Smalltalk), when a programmer creates an object, memory is explicitly allocated to it. When the object is not referenced by other objects anymore, that is to say it is not in use, a "garbage collection" mechanism deallocates or frees the memory that was used by that object. Hence, the programmer does not need to worry about memory deallocation, as would be required in procedural languages (such as C). [32]

Smalltalk is also furnished with a powerful debugger which allows the programmer to trace and control the flow of program execution. The debugger can be invoked when the user requests an interrupt for debugging, or an error occurs. In runtime applications when an error occurs, the user will be notified, but it is not possible to trace back the original reason which caused the error.

Chapter 4

Computer Automated and Integrated Design Philosophy

4.1 Introduction

The demand for expert designers has grown significantly because of the growth in demand for products. The Computer Automated and Integrated Design (CAID) system can make the knowledge of experienced designers more usable and accessible, and enable inexperienced designers to work more easily in the design process. This gives designers the opportunity to benefit from the exposure to more design information while allowing the design expert to undertake other creative and novel design problems.

During an engineering design process, an engineer designer rebuilds resources optimally to achieve a desired objective. The vast majority of solutions to engineering design problem already exist somewhere in the design domain. Finding these solutions and applying them to the design at hand is not so difficult.

One has to understand the process of design in order to develop a computer-aided design system. As mentioned earlier, engineering design is a process in which scientific principles, technical information and creativity are all employed in order to present a suitable final output which will function as it was proposed. In engineering, the design process begins with a unique stage which defines a specific problem and ends with specifying a suitable solution [7].

In this research a new environment is suggested to create a computer automated and

integrated design (CAID) system. This chapter will introduce The CAID and its philosophy.

4.2 Design Process

Although the stages of the design process differ from one engineering problem to the other, the order of these stages does not depend on the design problem. Hence several general patterns of the design process have been proposed. The CAID system uses a flexible design approach as will be seen in this chapter. The CAID system provides an appropriate environment for the designer in order to develop creative designs.

A design problem can generally be broken down into a number of stages, namely conceptual stage, and three main iterative stages, synthesis, analysis and optimization:

Conceptual Design: The conceptual design, which is the initial stage of any design, is typically stimulated by some need. This can be the demand for a certain device or a marketing department's proposal, or motivated by the needs of an experienced designer. The design process passes through the iterative stages that follow the conception of the idea.

Conceptual design is defined as the transformation from functional or behavioral requirements to structural descriptions. [42] For example, if the desired function is to transfer a rotation from one axis to another (and possibly changing the rotational speed), then possible embodiments are some shafts, gears, etc. Such a configuration design is defined as the transformation from a physical concept into a configuration with a defined set of attributes, but with no particular values assigned. For example, a gear train is a physical concept with the possible configuration of some shafts and gears.

As mentioned earlier, after the initial, or conceptual stage, the design process proceeds through the iterative stages of synthesis, analysis and optimization.

Synthesis: This stage is a primary characteristic of all successful design work. During this stage, design alternatives are developed. The only details accessible to the designer are the attributes of the end product. At the end of this stage, the designer specifies one, or at most, a few initial design alternatives that fulfill the principal constraints of the problem at hand.

During the synthesis stage, the designer studies the previous solutions and explores decomposing the system into subsystems. The complexity of these subsystems depends on the nature of the design problem. Each subsystem may be also broken down into simpler subsystems. At the end of this hierarchy will be the components or elements. For example, for the case of a gear train for reducing the speed of a shaft, the designer can assume a system with two shafts and a pair of gears. Each subsystem can be synthesized by considering all feasible combinations of its different components. Applying this hierarchical technique to the synthesis of a design problem allows the designer to consider a complete set of alternative designs. In this stage the design feasibility will be initially estimated.

Using the object-oriented approach to implement the CAID gives the system the ability to provide generic elements which are necessary to built any device. These generic elements, which are building blocks for the design in hand, will be represented by names so the element can be created just by pointing to its name, which would be added to the layout of the design in hand. This is like picking up a generic element from the shelf of the elements (e.g. spur gear). Similar to its real counterpart, this generic element contains all its attributes within itself (i.e., geometry, load, material

attributes).

In this stage, using this method, the designer will easily create the layout of the design. The information on the previous designs' layout could help the designer in the creation of the new layout. The creative part of this layout is the responsibility of the designer. During the creation of the design layout, the details on the layout are of no importance to the designer. The concentration will be mostly on the general organization of the design. This process will be illustrated through design examples in Chapter 8.

Analysis: The next iterative stage is the layout analysis. This stage concerns investigating the feasibility of the layout in more details. Mathematical and other scientific procedures are used in this stage to analyze a particular design alternative. In this way, the behavior of the design will be judged with respect to its applications. The selection of the appropriate analysis methods, the proper application of these methods, and the precise interpretation of the conclusions are very important in this stage. The hierarchical decomposition of the design in the previous stage would have reduced the complexity of the design by breaking it into perceivable sub-designs. In this way, the analysis of the design layout can be handled more easily. In the speed reducer example, the designer can determine the type of the gears, their diameters as well as the diameter and the length of the shafts based on the input speed, reduction ratio, and power transmitted. Furthermore, each subsystem will be independently described in detail. After the layout satisfies the constraints of the design, the designer starts parts development. Then the parts will be analyzed with the help of a new approach referred to here as a Part Design Diagram (PDD). This diagram is a model of Fuller-Polya Diagram which is described by G.Kardos in his paper "Problem Solving with the Fuller-Polya Diagram" [20].

This new approach will be discussed in detail in Section 4.2.1.

A subsystem must be interpreted by a model compatible to analytic or experimental measures before it can be analyzed. The designer must attempt to express as many of the fundamental characteristics of the actual system as possible in order to conceptualize such a model. This attempt will be restricted by the available time, methods, and analytic or experimental techniques. Conventional models are simplified actual forms, mathematical models, free-body diagrams, mechanical electrical analogues and models based on nondimensional equivalence. [8]

Optimization: Although the two first stages are sufficient for obtaining a feasible solution of a particular design, a final, or optimization stage is necessary to achieve optimal solutions. In the above example, after the first analysis stage, the designer will most probably run over the iterative cycle to obtain more improved design by applying optimization techniques. This cycling proceeds until the designer is confident that the design is optimal. The CAID system allows the designer to activate multiple design cases simultaneously; in this way, handling iterative stages is easier.

The iterative stages support the designer with the capability to optimize the design at different levels. Global optimization methods could apply on the design as a whole as the designer cycles through various stages. This will enhance and improve the design at the product level. Local optimization methods could apply on completed parts between the successive layout phases. This will enhance and improve the design at the part level.

The design fully matures as it progresses from a conceptual design to a complete design. At the end, a feasible, acceptable or optimal design solution to the specified problem will be produced. The design process starts from an abstract model and ends with a physical product or system. The

abstract models are described by parameters and charts while a maturing physical design is usually characterized by an actual product. When a particular design achieves maturity and the design is accepted, the design solution is completed. Furthermore, this solution may be used as the starting point for future designs.

4.2.1 Part Design Diagram (PDD)

Parts design is primarily a single solution deterministic problem. The use of the PDD increases the efficiency of the deterministic solution of the part design. Figure 4.1 shows a PDD diagram. This diagram shows the relations between the variables and the design equations or

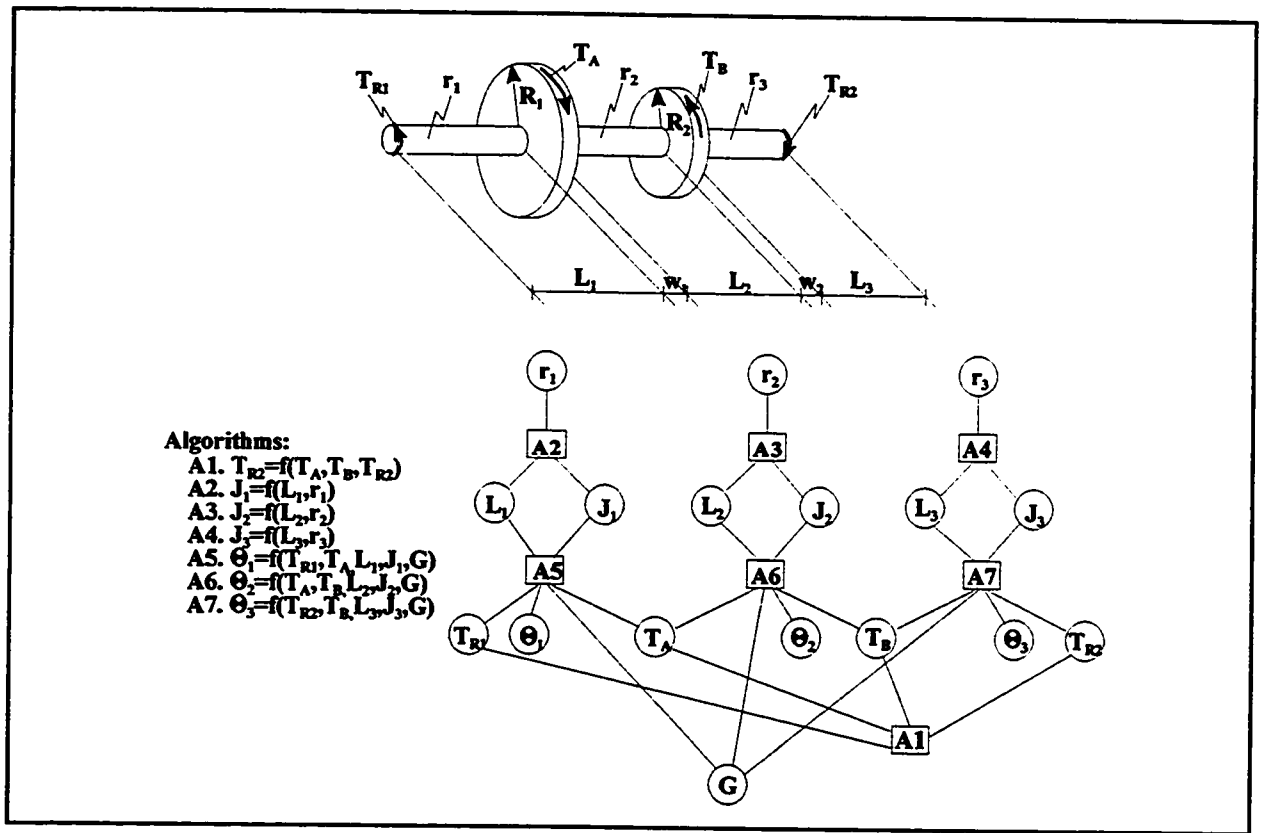


Figure 4.1 : Shaft Diagram with Applied Forces, Shaft Parameter Relation Algorithms and Part Design Diagram of Shaft

algorithms. In the implementation of this diagram unknown variables must be identified together with the physical principles and mathematical algorithms.

The diagram contains variables which may be known or unknown, and simple algorithms which are procedures that can be mechanically carried out without creative intervention. The variables are identified by a letter or a name, and appears only once. The algorithms are identified by letter 'A' and a number, and each algorithm acts as a connecting element of its variables. The diagram will be completed by joining the algorithms together through their variables. The variables are connected only to algorithms and algorithms only to the variables. Algorithms may have different forms; they can be mathematical equations, tables or graphs (e.g., shear diagram). It is not necessary that all the algorithms be explicit. This diagram will be used in the part design of the design model which is discussed in section 4.3.4.

4.2.2 Object Relation Diagram (ORD)

Figure 4.2 shows the object relation diagram. The diagram portrays the relationships between the elements of the design. This diagram can be derived by a physical decomposition of the system and finding out the relationship among the elements of the design. The ORD serves as a means of communicating design information between the designer and the CAID system. Each node of the ORD is an element and has its own part design diagram. In Figure 4.2 these PDD's are hidden.

With this diagram, each gear in the gear train is related to its gear mate, and each shaft recognizes its supporting bearings. The pointers to the gear mate or supporting bearings are not the only information placed in the connection. The exchanging force and torque between connected components are part of the connection and will be discussed in Chapter 6.

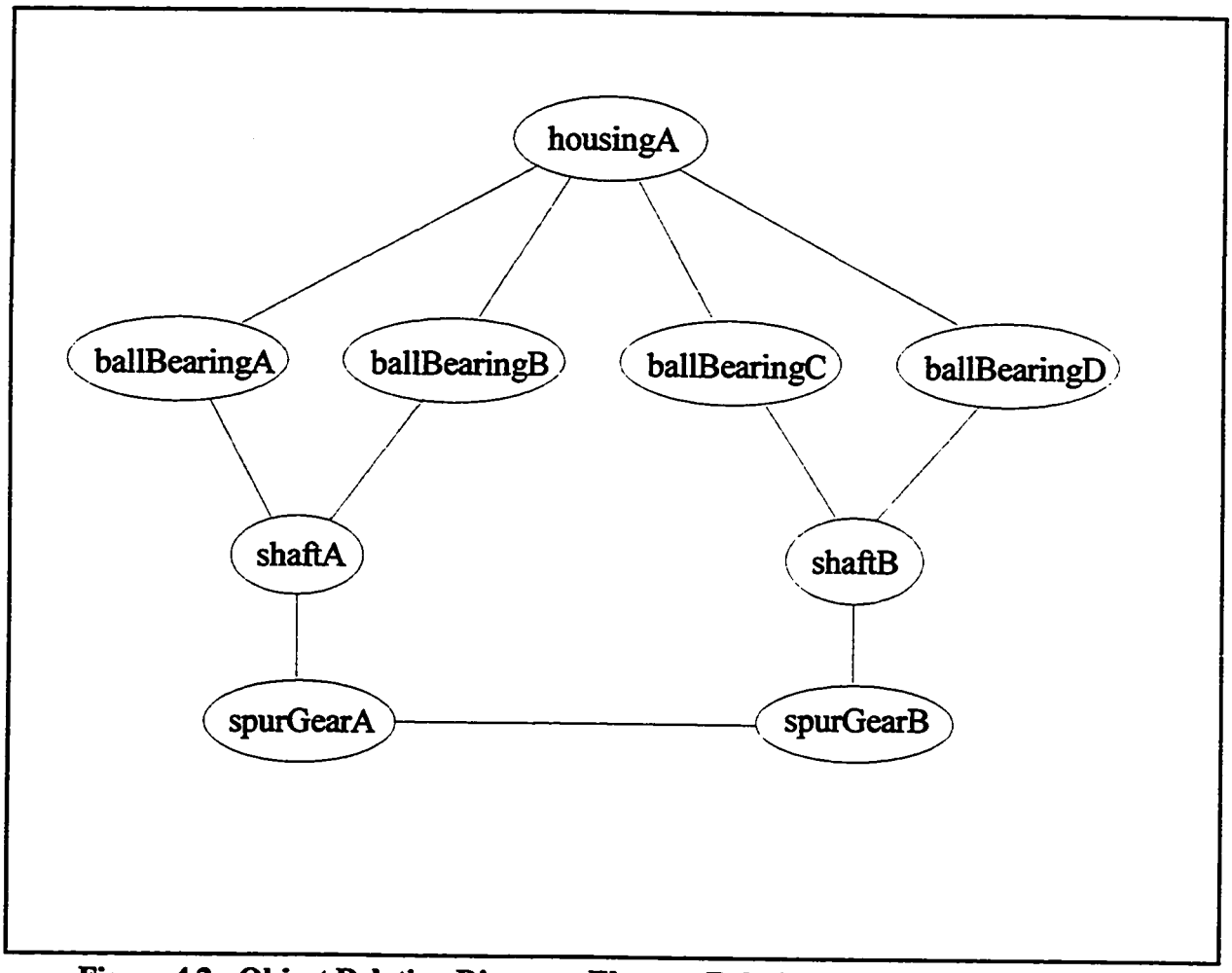


Figure 4.2 : Object Relation Diagram, Element Relations in a Gear Train Design

4.3 CAID Concept

CAID provides a design base on which a designer develops a design. Figure 4.3 shows CAID components and their relations. These are: Element base, Connection base and Design Knowledge base, Design Engine, Design Model and its instance Design Case.

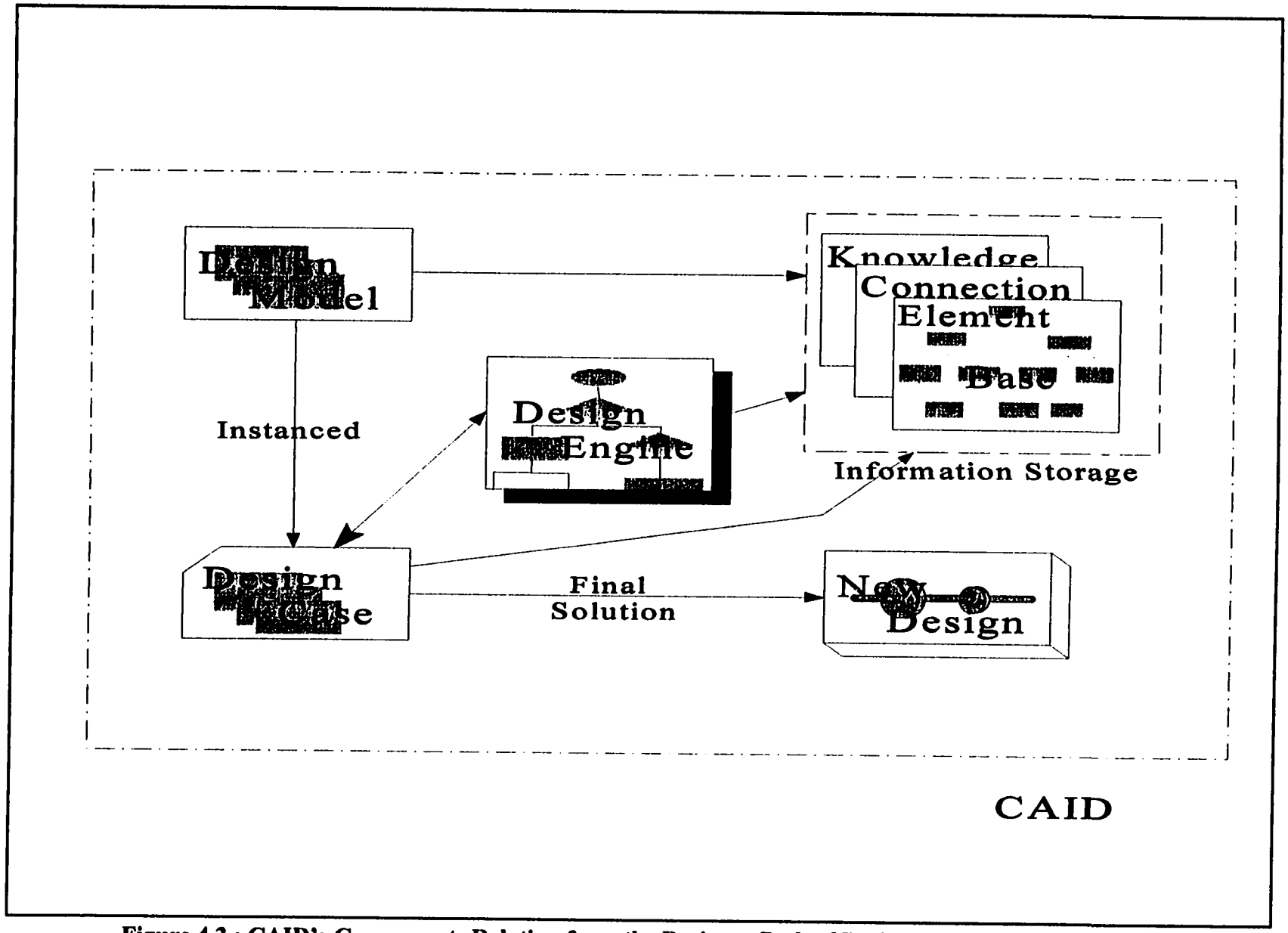


Figure 4.3 : CAID's Components Relation from the Designer Path of Design, Ending to New Design

In order to have a flexible and integrated design base or environment, a modular building block approach is used. Hence, the design knowledge is implemented as modular building blocks which are "methods" in the CAID's classes and will be explained in Chapter 6. This will allow integration of design tools, and make it easy to replace an old design tool with a new one without any side-effect on other parts of the CAID system. The building block approach is also used in implementation and organization of elements in mechanical engineering design. This approach contributes expandability and growth potential to the CAID system. Creating different abstract classes for different design models has made it possible to process the designs in various fields. This also gives the designer the power to process several design cases (instances of design models) simultaneously. In this approach, each design case is an integrated component in the CAID system which gives the designer the ability to switch between various design cases. Having the whole design case as an integrated entity makes it possible to store and retrieve them to be used as a starting point in future designs. There is a design engine which controls the design process while itself being supervised by the designer.

The CAID is organized in such a manner that the expert knowledge stored in it could be easily retrieved by the designer. The experience and the knowledge of the expert are placed in the three bases, namely Element base, Connection base and Design Knowledge base. For example, for the gear train design, the knowledge of the expert designer in the gear train design will be stored in the CAID system in these three bases. In the development of these bases the advantages of the object oriented systems are used.

Detailed description of the components of the CAID system follows:

4.3.1 Element Base

Elements are designable entities which form the basic building blocks of any design. The understanding of these smallest parts of the design and their structure supersedes the understanding of the CAID system. Elements are tangible entities the behaviors of which are different from various design views. Spur gears, helical gears, bearings, shafts and keys all could be assumed as mechanical elements. Attributes of these physical elements, such as size and mass, are not known in the Element base, rather they are "generic". Hence, the elements of each design would be defined as classes in the object oriented systems. These attributes will be defined or become "specific" only when the element is used in a design case. That means a copy or instance of the class of specified element will be created.

"Parameters" are structural properties of the element, which are instance variables in object oriented systems. For example, a gear may have the parameters of diameter, face width and number of teeth, while a helical spring may have the parameters of spring diameter, wire diameter, length, material, and number of coils. Whenever an instance of an element is created in the design case, the values of its parameters (which are the instance variables of the element) should be assigned or calculated.

As mentioned earlier, a copy of the element class is used in the design. In the design model, the element class is referenced and afterward, in the design case, when design takes place a copy or instance of the element is used. This copy which is an instance of the element class is called a component. It should be noted however, that in object oriented terminology this copy of an element class is called an object. Elements base will be defined as a hierarchy of classes which is described in detail in the next chapters.

4.3.1.1 Parameter Classification

Parameters of each element can be viewed from different points of view. From inside the element, its parameters can be dependent or independent. Independent parameters are not related to other parameters of the same element. Dependent parameters, on the other hand, have their value described by the relationship of at least one other parameter of the element. For example, the torque applied on a gear depends on the transmitted power and angular speed of the gear. The relationships of the dependent parameters to the other parameters of the element are described by functions. These functions can be mathematical or heuristic equations (e.g., obtaining Lewis form factor according to the gear type and numbers of teeth) [21], tables (e.g., finding basic stress for a gear according to its material) and the like. Each of these parameters may have more than one function to describe its relationship to other parameters. The designer could choose the function which will be used to calculate the unknown parameter according to desired accuracy, time for evaluation, and number of unknowns. For example, selecting a material for the gear in the first iteration, the designer could use the rule of thumb or a simplified table. In second iteration, if the gear size becomes too large, according to the desired gear size and the calculated induced stress on the gear tooth, the designer could select stronger and more suitable material for the gear. Finally, using the finite element method, the designer can analyze the gear stress in detail, and choose proper material accordingly. The different approaches for calculating of these parameters will be discussed in detail in the following chapters.

From the connectivity point of view, mechanisms in general have geometrical parameters that are common or related between elements of that mechanism. For example, the inner diameter of a gear is related to its shaft outer diameter, or the angular speed (rpm) of a gear is related to its

connected gear's angular speed and the number of the teeth of both gears. These kinds of parameters are called interface parameters. The interface parameters are indirectly accessible to other connected components, while the other parameters are not accessible.

There are some elements in each system which are related to outside of the system. The parameters of these elements which are related to outside of the system are called input or output parameters. For example, horsepower (hp) and angular speed (rpm) are input parameters of a gear train. These input parameters are related to the other components' parameters through interface parameters. The values of the input parameters are determined by the designer.

There are two different types of output parameters. The first types are relevant to the components of the system. The values of these parameters can be used in directing the design of the other connected components. Some of these output parameters are interface parameters, hence they will be used to obtain the interface parameters of the connected components. The second types of the output parameters are relevant to the system as a whole.

4.3.2 Design Knowledge Base

In the CAID system, design knowledge is extracted and organized in a base called the design knowledge base. There are two kinds of knowledge. The first pertains to how to design a part or component of the system. This knowledge is built into the CAID system as a part design diagram (PDD) for each element class. Part design diagram will be elaborated on in chapter six. The second kind is the knowledge of designing a system composed of several of the components. This knowledge is built into the CAID system as procedures (methods) in each Design Model class.

A "building block" approach is used for setting up both parts of this base. The designs are

broken down into small easily understood and described blocks that are defined as class methods. Typically the knowledge which is placed in the design model consists of rules (e.g. if angular speed is high use helical gear), heuristics (e.g., when the calculated gear ratio becomes more than 20 in a pair of spur gears, then the designer should use more than one pair.), mathematical (e.g., induced stress on the gear tooth (s_{in}) are mathematically related to: force applied on tooth (W), face width of gear (F), gear circular pitch (p), and Lewis form factor (y), [s_{in}= w/(Fpy)]) relationships. This part of the base also includes the design case solution procedures, or decision making methods. The knowledge which is placed in each component in its part design diagram consists of physical laws and properties (e.g., calculating the torque of a gear according its transmitted power, hp, and its angular speed, rpm) that handle components' design within a design process.

The information developed in one specific design case can be used in other design cases. This will eliminate reproduction of an entire similar processes, and will eliminate the duplication of knowledge if the design cases are stored in such a way that the knowledge of each design case is accessible to other design cases. For instance, a developed part design diagram of a spur gear in a transmission system could be used in other type of gear trains. Hence, if the design of a spur gear was required at a later time, for example throughout a planetary gear system design, there would be no need to develop the design process of the spur gear. This process has already been developed in designing the transmission system and stored in the part design diagram of the spur gear. The same process can be used in designing a spur gear of the planetary gear system.

Using inheritance, a part design diagram of a particular element can inherit most of its attributes from its superclass, and then only specific attributes of that part class would be added. All algorithms that apply to different kind of gears will be implemented in the Gear class. For instance,

if the gear design was based only on the teeth strength, there would be ten different algorithms to process a gear design. These algorithms and their variables, shown in Figure 4.4, will be added to the PDD of the Gear class with necessary connections between them. Later, by adding steps required by a particular gear design process (e.g., a helical gear design) the part design diagram of that gear can be completed. For example, in helical gear the velocity factor, C_v in the algorithm A6, has a new relation as: $C_v = 78/(78+\sqrt{V})$, hence, algorithm A6 is overridden in HelicalGear class. Also in algorithm A2, the helix angle, psi (ψ), effects on the value of the tangential tooth load which is 2t/d in the regular gear. Hence, a new A6 algorithm is implemented in HelicalGear class where a message is sent to its superclass (Gear class) to execute its A6 algorithm to calculate induced stress as the gear is regular. Finally dividing this value by tan, the induced stress of the helical gear will be obtained. Using inheritance approach facilitates creation of a gear family such as spur, bevel, helical and worm gears' PDD.

A1: $t = 63030 \times hp / rpm$

A2: $S_{in} = 2 \times t \times (P^3) / (k \times \pi^2 \times nt \times y)$

A3: $S_{all} = C_v \times S_o$

A4: $V = \pi \times D \times rpm / 12$

A5: $y = fun\ (nt, tt)$

A6: $C_v = 600 / (600 + V)$

A7: $S_o = fun\ (gm)$

A8: $Soy = S_o \times y$

A9: $D = nt / P$

A10: $F = \pi \times k / P$

Figure 4.4 : Gear Design Algorithms

4.3.3 Connection Base

The connection base, unlike the element base which deals with the internal aspects of an element, consists of information about the external links an element has with other elements in a

design. Interdependencies between linked components are important in the design process as when two gears mesh or a shaft is supported by a bearing. This base provides a better understanding of these interdependencies.

The connection base can be a separated base, or the information it contains can be embodied partly in the element base and partly in the design knowledge base. In the second case some instance variables of each element class will be introduced as interface parameters or variables, and some methods of that element class will represent this connectivity. Hence, the interfaces would have to be a property of the element it represents. In this case the element classes are not as "generic" as in the first case, because an element in different situations has various interface parameters. For example, a shaft supported by a ball bearing, has a different interface parameters compared to when restricted by a key. So, indication of active interfaces is necessary as not all interfaces are used in the design at hand.

Although connections are not tangible objects, physically separating the connection base from the element base helps understanding these two different entities (element and connection). Hence, in this research, the first case is assumed, which means the connection base will be a separated hierarchy. The connection hierarchy will be discussed in more detail through out other chapters.

4.3.4 Design Model

The Design Model is one of the most important parts of the CAID organization which should be elaborated on. Figure 4.5 shows detail of a design model. The design model is in essence an abstract model. A design case is a working copy of the design model, and is the one used for actual design. The design case is discussed later in Section 4.5.1. Both the design model and design case

pertain to the design process of a particular product and contain its necessary knowledge and procedures.

With reference to Figure 4.5, when considering the design as a whole, there are components which are related to the outside of the system. These kinds of components are called input or output components and, as mentioned earlier, their related parameters to the outside of the system are called input or output parameters. The values of the input parameters are determined by the designer, and the values of the output parameters will be determined throughout the design process of the system.

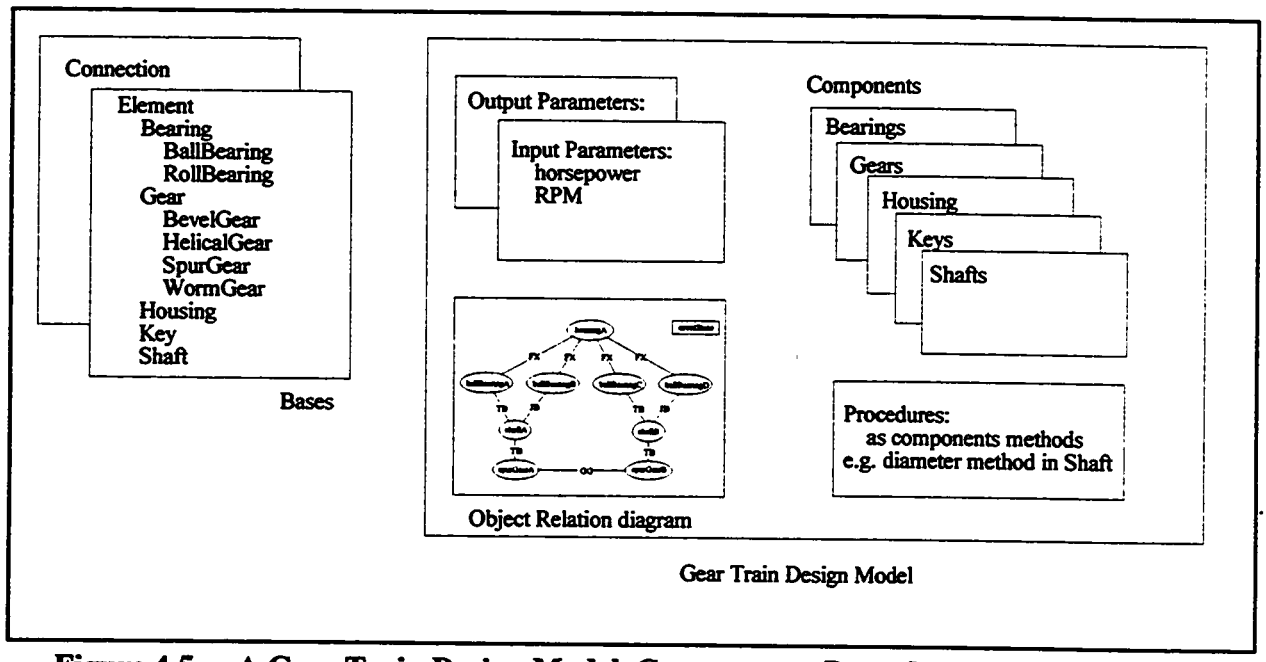


Figure 4.5 : A Gear Train Design Model, Components, Procedures, Connection and Element Bases

The output parameters describe the final design.

An individual design model should be created for each type of product. Each design model comprises necessary components, their connections (Object Relation Diagram), design procedures, and input and output parameters. The different design models, which represent different products

design, can be created using a general class namely DesignModel.

Components as fundamental parts of a design model are instances of the Element subclasses. Throughout the process of the design model of a particular product, copies of these subclasses with real parameters will be created. This is like selecting an element (e.g., a gear) off the shelf. The difference is that, during the design process, the designer can change the attributes of the element (e.g., gear diameter).

Procedures are simplified algorithms implemented in the design model. These algorithms direct the computation of required data throughout a particular design process. According to the results of the previous procedures, the proper decisions are made. Determining the diameters of meshing gears based on the gear ratio is an example of the procedure placed in the design model of the gear train design. Based on the results of this procedure and the stress applied on the teeth of those gears the appropriate material for the gears can be chosen.

In order to start the design process in a design model, there must be some known input parameters. For example in a gear train design model, input horsepower, input speed and desired output speed are input parameters, and should have known values before starting the design process.

4.3.5 Design Engine

The CAID system requires a mechanism to drive the design process. This mechanism is the Design Engine. It controls and directs all stages of the design process. In order to complete a particular design, the design engine follows the instructions implemented in the design model. It also provides feedback from the expert's knowledge to the designer. The designer has control on the design engine decision making process. For example, knowing transmitted horsepower, input and

output shaft's speed, the design engine will help the designer to complete a proper gear train design using the expert's knowledge stored in the CAID system.

Design Engine is not a class by itself; it is a set of methods which are implemented in DesignModel, Element, Connection, ORD and PDD classes and their subclasses. These methods with their collaboration direct the design process and communicate with the user. For example, the method copyDesignModel in DesignModel class creates a new copy from the receiver, being a specific design case. Hence, the user can have different copies from the same design case, and process them in parallel. Also gearToGearForces and shaftToGearForces methods in GearTrain class calculate the forces exchanging between meshed gears or shafts and their mounted gears respectively.

4.4 Design Solution

As mentioned, to find a proper solution for a particular design, the knowledge engineer extracts the design knowledge from the senior designer and develops a design model which consists of elements and connections along with the related procedures and algorithms. The elements and connections in the design model are only pointers to already defined elements and connections in the element and connection bases. If there is a new element which is not already in the element base, it will be developed by the knowledge engineer.

The procedures and algorithms related to a specific design model should be developed in a way so it can be implemented in hierarchy for getting the best inheritance usage from object-oriented programing. For example, to implement the pump design in the CAID system, the senior designer should consider a general pump as superclass of all type of pumps with parameters which are

common among all types of the pumps and generalizes its design in a way that all those algorithms developed for this general pump can be applied to all type of the pumps. For example, the following equation will relate required torque to other parameters of a general pump [43]:

$$T\omega = Q\gamma H$$

where T is torque,

ω is rotational speed,

Q is flow rate,

γ is unit gravity force,

H is pump head.

By creating new subclasses to this general pump (e.g., centrifugal pump) with specific parameters for that kind of pump, the senior designer will then add specific algorithms which can be applied only on this type of pumps.

Gear train design is chosen to verify the ability of the CAID system. Gear train design is a good example for demonstrating the process of generalizing the elements and their design procedures. That is because there are different types of gears and bearings and they can be organized in hierarchy form. The generalization of the gear train elements and their force analysis are discussed in Chapter 5.

The junior designer uses the design engine to make a working copy of the design model called the design case. Although the design engine manages the design case, the designer has complete control over the entire design process. This means that the design process, in any level or stage, can be automatic or guided by the junior designer. For example, in processing the part design diagram of a component, the junior designer can pass the control authority to the design engine to

complete the component design. In this case, the design engine will choose the order of the algorithms in which they should be executed. An alternative option for the junior designer is to choose an algorithm at a time based on results of the previous algorithm execution, and to change some parameter values.

In the beginning of the design, there will be a general design model. As shown in Figure 4.6, this design model consists of a title bar and five subpanes. The user can name the design model by pointing to this title bar. The five subpanes are called : list-pane, element-pane, graph-pane, message-pane and edit-pane. The Element subclasses will appear in the element-pane in inheritance hierarchy form, and the design system at hand with its subsystems will appear in the list-pane in ownership hierarchy form. Unnecessary Elements in element-pane can be hidden. There are two subclasses of the class Element which are called SimpleElement and CompoundElement, each containing different characteristics. Compound Elements, is a multi- Element containing ORD and a list of components, while Simple Elements have PDD.

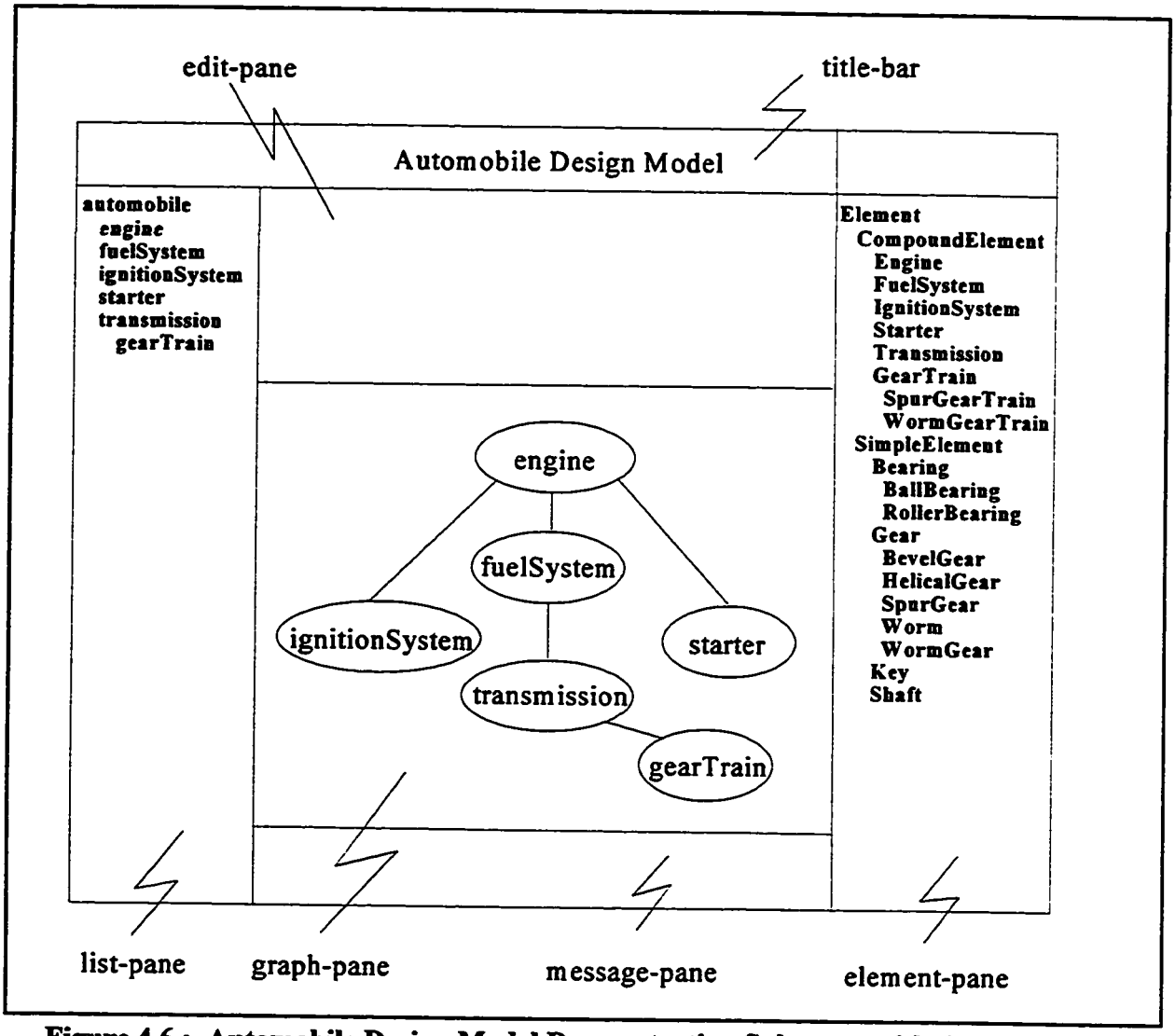


Figure 4.6 : Automobile Design Model Demonstrating Subpanes with their Elements

Using the top-down method , the user defines a new system to design (e.g., automobile), then chooses its subsystems from predefined compound elements which are available to the user. Connecting these subsystems, which are components of the main system, the ORD of the system will be created. The user can choose a subsystem in any level of the ownership hierarchy. Every time that the user selects a subsystem to design, a related design model of the subsystem will be opened. The components of the selected subsystem with the relation among them, which are represented by ORD

components of the selected subsystem with the relation among them, which are represented by ORD of the subsystem, will appear in the graph-pane. The user will continue the design process by designing the selected subsystem. During the design process of a subsystem, several copies of the subsystem design case can exist, but when the user wants to analyze the main design model of the system, only one copy of each subsystem can be considered.

With the aid of the design engine, the suitable solution will be completed. Evaluation and decision-making are done by the design engine based on the results of the design procedures. The designer regularly receives the feedback of all the evaluations and decision-making processes of the design engine, and finally decides to terminate or revises the design case.

4.4.1 Design Case

Once a design model has been created, a specific design case can be used as its "instance" with a given set of user design parameters. As mentioned previously, the knowledge engineer develops a design model, and as its name implies, it acts as a model to create a design case. The junior designer begins any particular design by creating a working copy, or instance, of the related design model, which is the design case of the particular design model. While the design model is generic, and just exemplifying the design in an abstract form, the design case deals with the developing of a specific design.

Referring to Figure 4.5, it is clear that the same concepts as in the design model are accessible in the design case, but it does not mean to copy all of them in the design case. For example, whenever one needs a procedure to be processed, instead of copying it in the design case, the design engine simply runs the same procedure in the design model with the new data and returns

the results. This is like a subroutine call in traditional programing or sending a message to a particular class of the design model in object oriented programming languages like Smalltalk. In design case, there are components which are instances of Element subclasses rather than elements themselves as in the design model. Connections are also instances of the classes of their own type. Instances of the Element and the Connection subclasses are objects in the Smalltalk, which means, components that are instances of the Element subclasses or design cases being instances of the DesignModel class are not abstract like their classes. A new design case needs memory only for storing its instance variables values. But there is no need to copy the reusable information like methods from DesignModel class to make its instance as a new design case. Hence, the memory required by several design cases is very small comparing to the memory the set of the methods in DesignModel class needs. Input and output parameters are the instance variables of the components which describe physical properties of the design.

Using the design case instead of the design model enables the designer to work simultaneously on multiple design cases. These multiple design cases which initiate from the same model can act as various extensions of a particular design. In this way, the designer is able to compare several different possibilities of the design, and to choose the best of them. In order to have multiple design case on the screen, it is better to create just one design case in the beginning of the design process. Then, whenever there are several feasible answers to some parameters, the design engine duplicates the design case. Keeping the number of the design cases reasonable, the deadlock and uninteresting design cases will be deleted.

4.5 CAID Users

The CAID system needs at least one person to supply the design knowledge, designated as Senior Designer, a second person to arrange the design knowledge, namely Knowledge Engineer and a third person to design, i.e., Junior Designer. These persons could be represented by a single individual but usually they would be different persons.

4.5.1 Knowledge Engineer

The knowledge engineer is the main organizer of the CAID system. If different from the senior designer, the knowledge engineer is responsible for feeding the design knowledge into the system. This knowledge is extracted from the senior designer in such a way that the information needed for a specific design process can be placed in the CAID system. That specific design area will become part of the CAID system expertise. In this way, the knowledge engineer can combine multiple expert knowledge and increase the CAID system's proficiency in different fields of design. While this base of knowledge and experience can be used by junior designers at anytime, the expert designer will have time to deal with creative and novel design problems.

The knowledge engineer creates a design model for each product. According to the knowledge extracted from the senior designer about designing a specific product, the knowledge engineer builds the ORD of the product and the connections of its components. It is possible that some elements, which are found in this product, are not already in Element base. The knowledge engineer, based on the knowledge extracted from the senior designer, implements a new class for these elements and their PDD. The knowledge engineer also defines the interface parameters of the components that relate them to other components in the product.

4.5.2 Senior Designer

The senior designer is chosen to be an expert in a particular design discipline. These senior designers or experts are major assets of design offices or groups. Transferring the knowledge of these experts to a junior designer is one of the main concerns of such design offices. Traditionally this transfer of knowledge has occurred throughout the process of training for new designers or through an apprentice program, either officially or unofficially. Presently there are some difficulties with implementing the traditional transfer of knowledge due to the generation and technology gaps, as well as the transient nature of current design office personnel.

As mentioned, the knowledge engineer will extract expert knowledge from the senior designer. This knowledge acquisition can be done by having an interview with the senior designer, or having that designer construct a design expertise file. Having extracted the knowledge of experts, the knowledge engineer could implement necessary bases and design models using the design expertise file. This approach allows the knowledge of the design expert to be used efficiently by others.

Another responsibility of the senior designer is to verify the contibution to the design by the junior designer using the CAID system. The senior designer's signature is required on any documents to be used in the manufacturing process.

4.5.3 Junior Designer

The junior designer who will often be referred to as a user or designer, is the person who uses the CAID system to complete a design. The knowledge held in the CAID system will be used by the junior designer through the design process. The junior designer can direct the design process by

having feedback from the design engine and have control over the whole process. This make the junior designer an active part of the design process. The CAID system helps the junior designer to acquire the skills and experience of the senior designer by actively being involved in the design process, and by observation.

4.6 CAID Accessibility

The knowledge engineer, the senior designer and the junior designer's access to the CAID system is shown in Figure 4.7. Each of them has access to different parts of the system because each role is basically different. The knowledge engineer is the only person who has access to the whole CAID system, and can extend applications of the CAID system by creating new classes or modifying existing classes using the Smalltalk environment. The Design Engine is the central part of the design system which has connection with most of the system components. The junior designer can communicate with the design engine and has access only to the design cases which are being processed. The junior designer creates the design cases with the assistance of the design engine. As mentioned, the junior designer is able to define new systems using predefined subsystems. Then these subsystems can be designed with the help of the design engine supported by the knowledge of the senior designer with the information stored in the CAID system. The senior designer, if different from the knowledge engineer, can use the CAID system like the junior designer to study the system's capabilities, and then create a design expertise file to give new suggestions and ideas for extending the system's abilities.

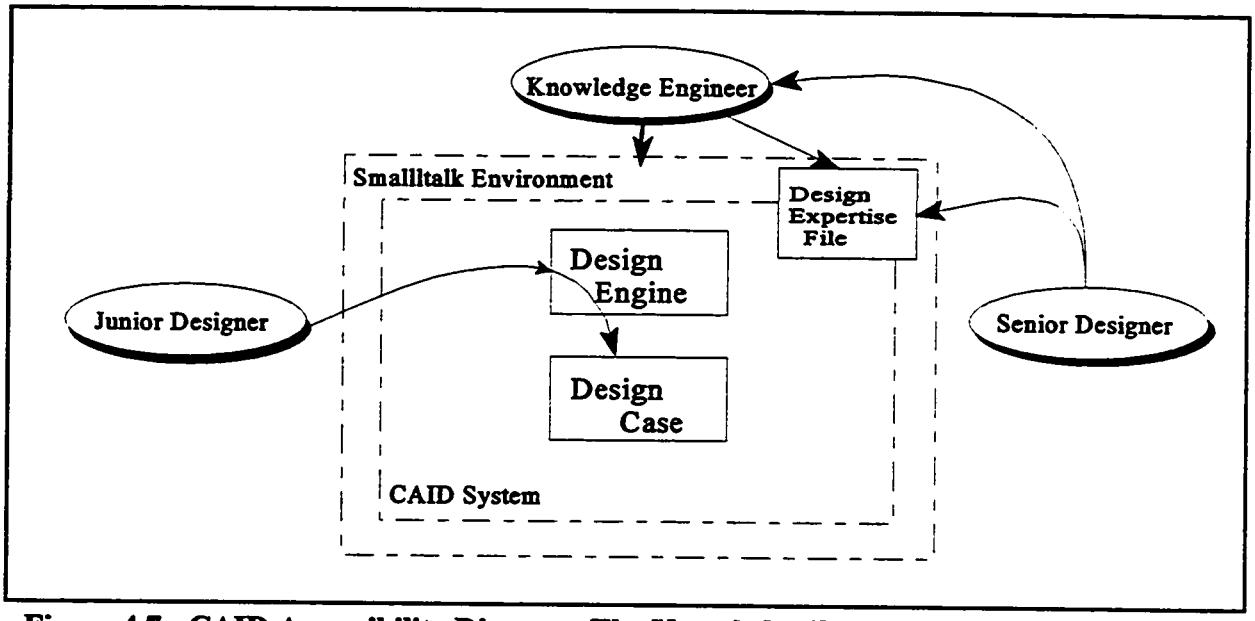


Figure 4.7 : CAID Accessibility Diagram. The Knowledge Engineer, Senior Designer and Junior Designer access different parts of the system.

It is possible to develop software to implement the knowledge stored in the design expertise file in the CAID, but this is not the aim of this research. Figure 4.8 shows the protection rings of the CAID regarding the accessibility of its users to different layers (levels) of the system.

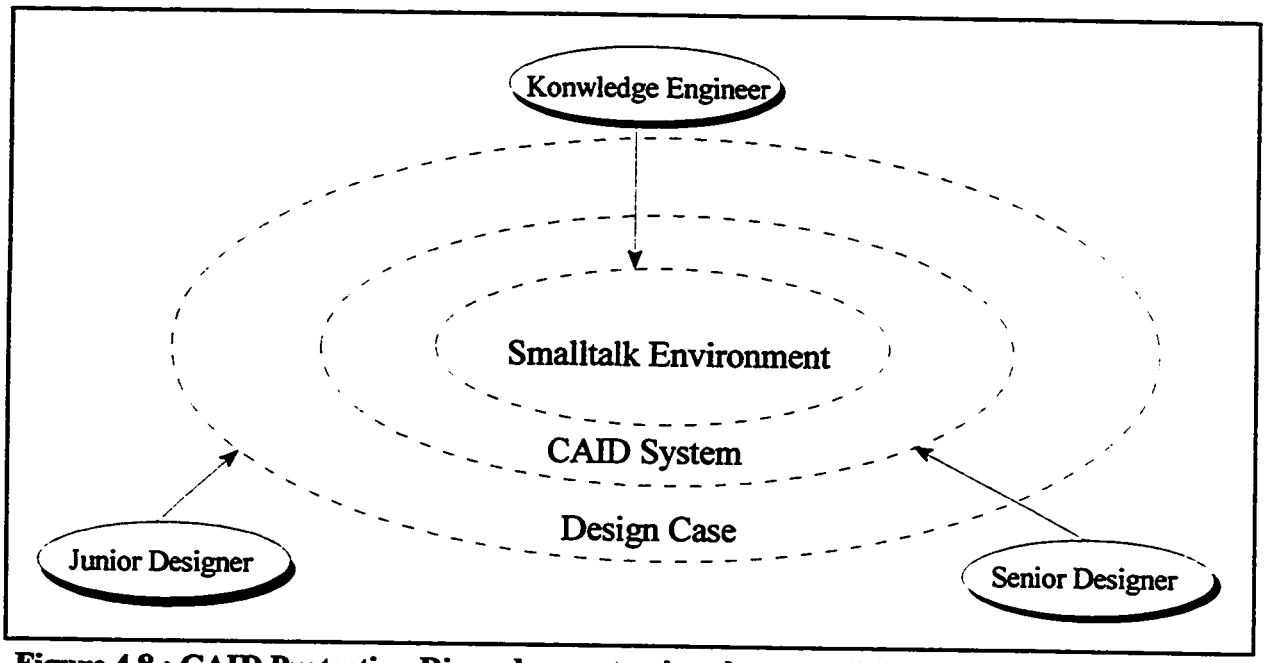


Figure 4.8 : CAID Protection Rings demonstrating the accessibility of its users to different layers of the system

Chapter 5

Theory of Gear Train Design

5.1 Introduction

Gear train design is still somewhat of an empirical art supported by scientific basis. Gear train design, like most mechanical engineering designs, is an iterative design; hence it does not have a unique solution and requires a trial and error method of solution. Such designs usually have more unknown parameters than equations. Hence, the process is usually carried out by making initial assumptions based on results from past experience. Some initial assumptions are checked in terms of standard values, proportions, or range in values.

Gear train design requirements are as follows:

1) sufficient gear teeth strength for static and dynamic loading;

2) good wear characteristics for satisfactory life;

3) economical use of space and material;

4) consideration of gears alignment and shafts deflections;

5) good lubrication.

The following specifications are the minimum requirements to start a gear train design:

1) the horsepower to be transmitted;

2) the speed of the input shaft, and

3) the speed of the output shaft or the velocity ratio.

Among the different components involved in the design of a gear train, gears and shafts are of primary importance. Hence, in this chapter, a comparison of the conventional method and the object-oriented approach for the gear and shaft design will be viewed.

The conventional design methods for gear and shaft are from three different sources, [21], [22] and [23]. These design theories are organized in a way that can be easily converted to the object-oriented approach in mechanical engineering design.

5.2 Gear Design

Gears, like belts and chains, are used in machines and mechanisms for transmission of power and for speed reduction. The American Gear Manufacturers Association (AGMA) has defined [21] gears as machine elements that transmit motion by means of successively engaging teeth. Although gear technology has advanced in recent years, gears have a very old history and were used throughout ages.

In this section, first the geometry of gear-teeth will be discussed; then the design of the spur gear, as a simplest gear, will be considered from a different point of view. Later, in Section 5.2.3, differences between the design of the spur gear and other type of gears will be discussed. The phase in which these methods are modified for the object-oriented approach, will be discussed in Section 5.2.4. Finally, forces that are exchanged between different type of meshed will be analyzed. The results are useful in designing the gear connection.

5.2.1 Gear-tooth Geometry

The exact constant angular velocity ratio is the prime consideration of the gear-tooth geometry design. The teeth of ancient gears, used by Chinese about 2600 B.C., were round wooden pines which did not satisfy this prime consideration. When the cast teeth gears were used for the first time, in their day, they were satisfactory regarding the constant angular velocity ratio.

A pair of gear teeth which satisfies this constant angular velocity ratio, are named conjugate gear-teeth. In 18th century, Euler described the principles of conjugate gears. Although the conjugate gear-teeth can have various shapes, the only one of current importance is the involute of the circle. An involute (of the circle) is the curve generated by any point on a tense string as it uncoils from a base circle (Figure 5.1).

The terminology applied to gears includes much nomenclature. Readers who are not familiar with these terms, should refer to Appendix B.

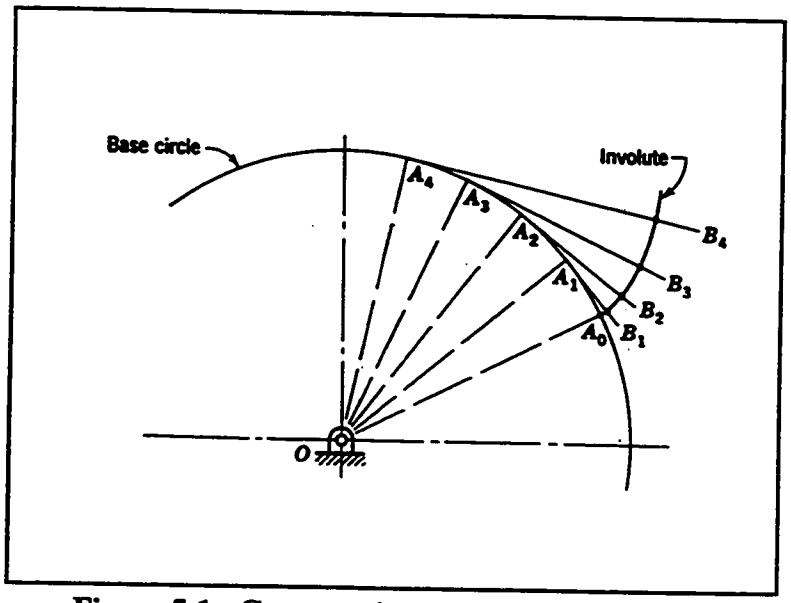


Figure 5-1 : Construction of an Involute Curve

5.2.2 Conventional Design of Spur Gears

In order to start a gear design, the designer generally needs the following specifications:

1) the horsepower to be transmitted;

2) the speed of the driving gear, and

3) the speed of the driven gear or the velocity ratio.

Frequently the center distances of some shafts are also specified.

In the design of gears for a good service, there are important items that must be considered. These items are: strength of teeth under static and dynamic loading, silence and smoothness of operation, available gear cutting equipment, proper lubrication, temperature and heat dissipation, and satisfactory life.

In the following sections only the simple methods in gear design are considered to be able to apply object-oriented features (e.g. inheritance).

5.2.2.1 Strength of Gear Teeth, the Lewis Equation

The determination of the maximum stresses in a loaded gear tooth is complicated because of the load variation on the tooth, in both magnitude and direction, and the variation of the gear width. W. Lewis made some assumptions which simplified the equation for obtaining the strength of the gear teeth.[21] The first assumption was that the worst position of loading the tooth occurred at the first point of contact and that one pair of mating teeth carried the entire load. The second assumption was that the gear tooth could be regarded as a cantilever and the principle that a beam of parabolic outline has uniform strength at all sections can be used. With these assumptions, the base section of the tooth (TC in Figure 5.2) will be the critical section.

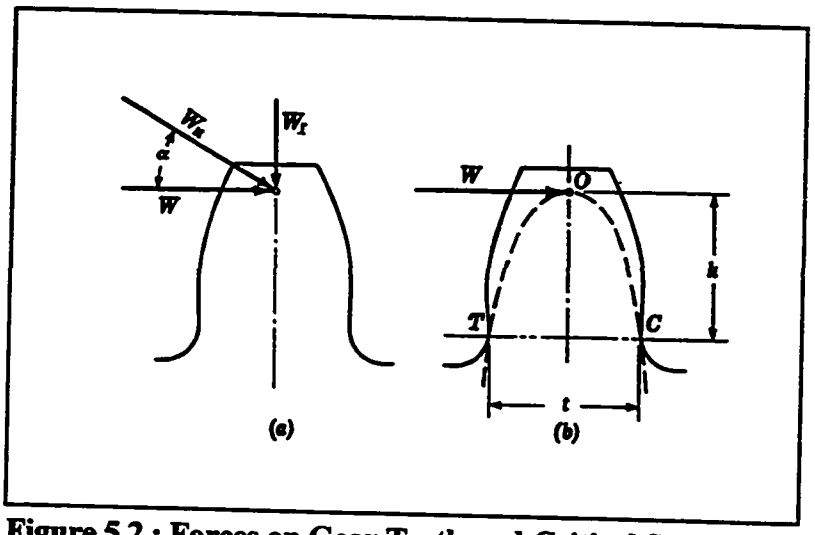


Figure 5.2 : Forces on Gear Teeth and Critical Section TC

The equation for tangential force, W, can be used to determine the proportions of the gear teeth. Applying the flexure formula (σ_m=Mc/I [24]) to this section with following relations:

$\sigma_m = s$, $M = Wh$, $c = t/2$, $I = Ft^3/12$

where F = the gear face width,

T = the gear width at the base section,

h = the height of the force W from the base section.

Results:

$s = Wh(t/2)/(Ft^3/12) \Rightarrow Wh = sFt^2/6$

Substituting Lewis form factor, $y = t^2/(6hp)$, results in:

$$W = sFpy. \tag{5.1}$$

The value of W may be calculated as the torque on a gear divided by its pitch radius. This value is somehow greater than real tangential force but it is good approximation. The face width, F, is usually made from two to five, or preferably, between three to four times the circular pitch for spur gears.

The Lewis form factor is a dimensionless quantity. If the gear is enlarged, the distances **t**, **h**, and **p** will each be increased proportionally; hence, the value of **y** will remain unchanged. The value of **y** then is independent of the size of the tooth and depends only on the number of teeth on a gear and the system of the teeth. The Lewis form factor can be obtain using different methods, analytically, graphically or from a table of precalculated values. Values for standard tooth forms are given in Appendix C.

Although most gear design equations are relatively easy to apply to an existing set of gears, they are not of such form that they can readily be used for direct design. The Lewis equation in its original form, or modified for a special class of gears or service, is a most helpful aid in first stage of gear design. In order to calculate induced stress on gear tooth, the Lewis equation can be written as:

$$s_{ind} = W/Fpy \tag{5.2}$$

For the solution of problems in which the center distance and velocity ratio are specified, this equation can be written as:

$$s_{ind} = W P^3 /k\pi^2 y \tag{5.3}$$

where F replaced by k.p, k = 3 to 4 for ordinary service, and

p replaced by π/P

p is circular pitch

P is diametral pitch

In the case the center distance is not specified, induced stress can be expressed as:

$$s_{ind} = 2TP^3/k\pi^2 ny \tag{5.4}$$

where W is replaced by 2T/D or 2TP/n

Since there is not a unique solution, the final design will depend on the initial assumptions. The quantities to assume should be those for which judgment is most easily exercised.

A primary concern in gear strength is that of fatigue due to bending. Most gear teeth are loaded in only one direction. However, the teeth of idler gears and plant pinions are loaded in both directions. The effect of load fluctuation on these kind of gears (Figure 5.3) should be considered in their design process.

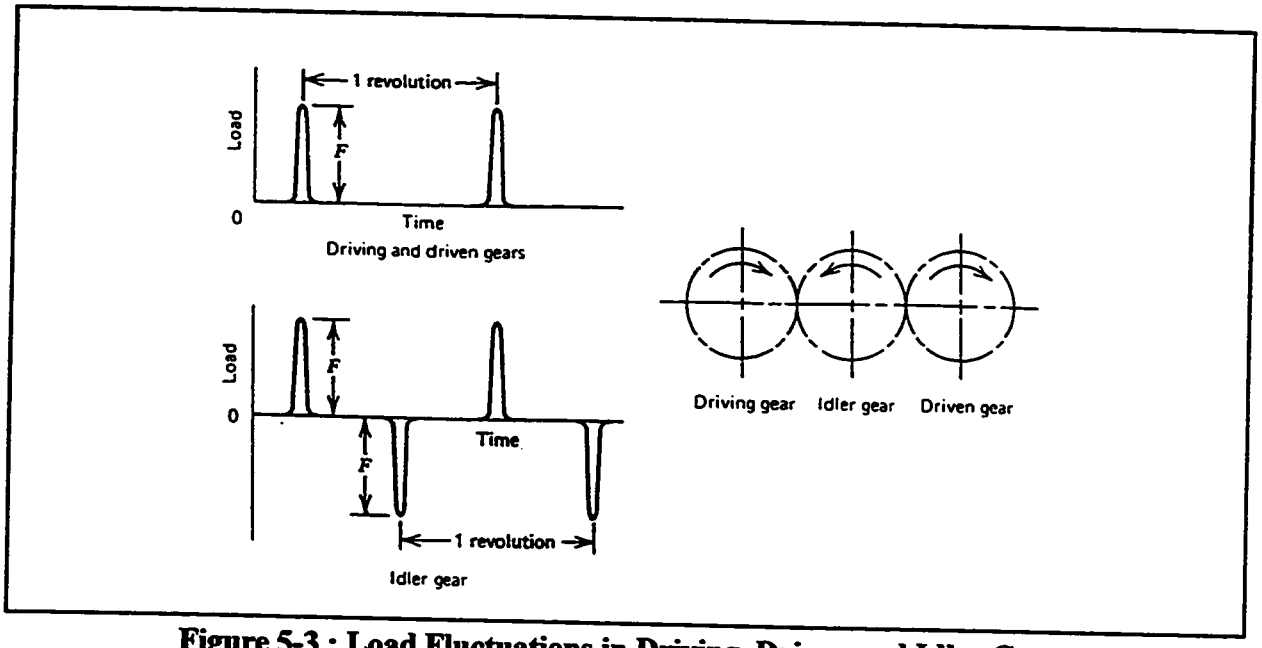


Figure 5-3 : Load Fluctuations in Driving, Driven and Idler Gears

In order to make allowance for the dynamic effects due to the velocity of tooth action, the Barth equation [21] may be used, i.e.,

$$s_{all} = s_o C_v \qquad (5.5)$$

where s_{all} = allowable stress, psi

s_o = basic stress, psi

C_v = velocity factor

= 600/(600 + V) for spur gears

V = pitch-line velocity, fpm

5.2.2.2 Dynamic Load on Gear Teeth, the Buckingham Equation

The dynamic load concerns are due to three factors: inaccuracies of tooth spacing, irregularities in tooth profiles, and deflections of the teeth under load. Based on an extensive series of tests [25], the dynamic loads on gear teeth, such as those in Figure 5.4, can be approximated by following equations:

$$W_d = W \pm W_i \qquad (5.6)$$

where W_d = total dynamic load

W = steady load due to transmitted torque

W_i = increment load due to dynamic action

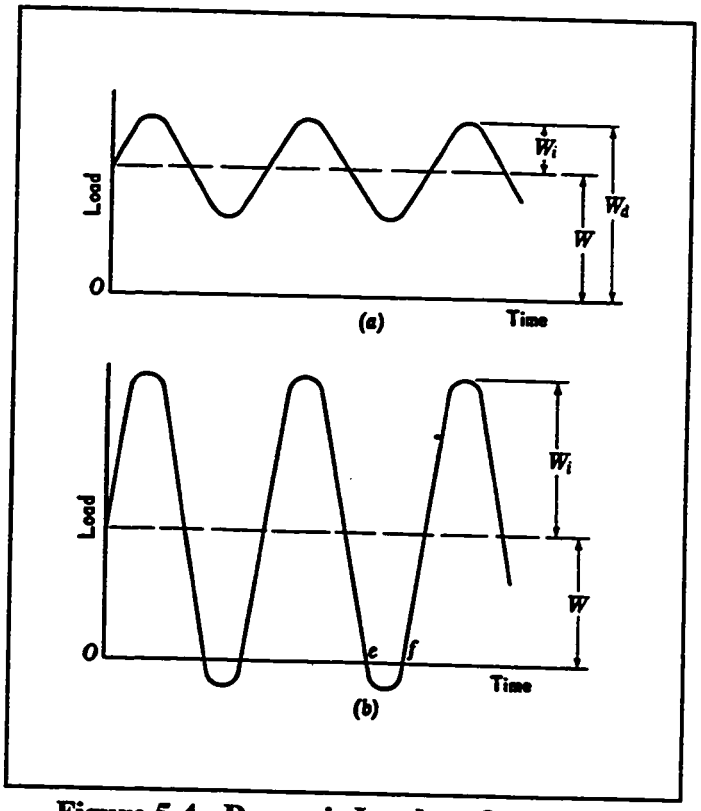


Figure 5-4 : Dynamic Load on Gear Teeth

The increment load W_i depends on the pitch-line velocity, the face width, the material of the gears, the accuracy of cut, and the tangential load W. For average conditions of mass of the gears and connected parts, the following equation applies [21]:

$$W_d = W + (0.05V(FC+W)/(0.05V+\sqrt{(FC+W)}) \qquad (5.7)$$

where W_d = maximum dynamic load, lb

W = steady transmitted load, lb

V = pitch-line velocity, fpm

F = width of face of gears, in.

C = deformation factor

5.2.2.3 Wear of Gear Teeth, the Buckingham Equation

Wear is the most common failure of a gear. The maximum load that gear teeth can carry without premature wear depends on the radii of curvature of the tooth profiles and on the elasticity and the surface-fatigue limits of the materials. Buckingham found the following equation for estimating the maximum or limiting load for satisfactory wear of the teeth.

$$W_w = DFKQ \tag{5.8}$$

where W_w = limiting load for wear, lb

D = pitch diameter of pinion, in.

F = face width of gears, in.

Q = ratio factor

= 2r/(r+1) for external gears

= 2r/(r-1) for internal gears

r = velocity ratio

K = load-stress factor

$= (s_{es}^2 \sin \phi)/1.4\ (1/E_1 + 1/E_2)$

s_{es} = surface endurance limit, psi

ϕ = pressure angle

E_1, E_2 = moduli of elasticity of materials, psi

5.2.2.4 Geometrical Parameters

Two tooth forms for spur-gear teeth are standardized by the AGMA. They are full depth with pressure angle 20 or 25 degrees. Their addendum are 1/P and their dedendum are 1.25/P. It is important that a satisfactory tooth form be chosen for each application. In general, quietness is favored by the low-pressure-angle forms, but they are weaker than the high-pressure-angle ones. Gears with up to 20 diametral pitch are called "coarse-pitch" and gears with 20 or more diametral pitch are called "fine-pitch". Fine-pitch gears are used primarily for transmitting motion rather than power.

The face width of spur gear, **F**, usually is 3 to 4 times of its circular pitch, **p**. The pitch diameter of a spur gear is related to its circular pitch and number of teeth by the formula:

$$D_g = N_g\, p/\pi. \tag{5.9}$$

In some spur gear design, center distance are specified. In this case, following relations will be useful:

$$C = D_g + D_p \tag{5.10}$$

where C = center distance

D_g = gear pitch diameter

D_p = pinion pitch diameter

$$\omega_g/\omega_p = n_p/n_g = D_p/D_g \tag{5.11}$$

where ω = angular velocity

n = number of teeth

D = pitch diameter

Subscript "g" represents gear and subscript "p" represents pinion. So by knowing values of center distance and velocity ratio, the pitch diameters of the gear and the pinion can be determined.

If the gear is to be keyed to a shaft, the maximum permissible bore should be as follow:

$$d = D(0.50 + 0.0344 \sqrt{(n-12)} \qquad (5.12)$$

where d = maximum permissible bore

D = pitch diameter

n = number of teeth

The equation allows for a standard square key with the centerline of the keyway in line with the centerline of a tooth space. This is the best location for the keyway. This applies to gears with 12 to 24 teeth. For gears with more than 24 teeth, n = 24.[21]

5.2.3 Other Gear Design

As mentioned in Section 4.4, the senior designer should create a superclass for all type of gears, with parameters which are common among them. Then the gear design should be generalized in a way that allows for their common procedures to be implemented in that superclass of gears. Then procedures for specific gears would be determined by algorithms. So far, there were no attempts to generalize the gear design in a way that would be applicable to all gear types in different applications.

5.2.3.1 Helical Gear

Because of the gradual engagement of the teeth, helical gears run more quietly than spur gears and may be operated at pitch-line velocities up to 10000 fpm and over. High pitch-line

velocities mean low tooth loads. The value of ψ, helix angle (Figure 5.5), is not standardized, but because of end thrust, commonly ranges between 15 and 30 degrees. In order to balance axial loads, two pairs of helical gears or herringbone gears are used. In this case, the value of ψ can be up to 45 degrees. When helix angle = 0 then the gear is a spur gear.

According to the AGMA, by some modifications to the equation which are used to design a spur gear, can also apply to helical gear design. The Lewis equation will be modified to:

$$W = s_o FpyC_v/C \tag{5.13}$$

where W = tangential tooth load, lb

s_o = basic stress, psi

p = circular pitch in plane of rotation, in.

y = form factor; pressure angle is in plane of rotation

V = pitch-line velocity, fpm

F = face width, parallel to axis, in.

C = wear and lubrication factor;

= 1.15 for continuous lubrication,

= 1.25 for scanty lubrication,

= 1.35 for indifferent lubrication.

$C_v = 78/(78+\sqrt{V})$

Using a modified equation of Buckingham, the surface durability of the helical gear teeth can be determined as follow:

$$W_w = DFKQ/\cos^2\psi \tag{5.14}$$

The dynamic load on helical gears may be estimated from an extension of the dynamic load equation which applies to a spur gear by substituting $\mathbf{F \cos^2 \psi}$ for **F** and by multiplying the last term (increment load term) by $\cos \psi$ as follow:

$$W_d = W + (0.05V(FC \cos^2 \psi + W) \cos \varphi/(0.05V+\sqrt{(FC \cos^2\psi + W)}) \tag{5.15}$$

For gradual transfer of the tooth load the minimum face width for helical gear should be:

$$F \geq 2p/\tan \psi \tag{5.16}$$

As with a spur gear, the pitch diameter of a helical gear is related to its circular pitch and number of teeth by the formula:

$$D_g = N_g\, p/\pi. \tag{5.17}$$

The velocity ratio of any type of toothed gearing is equal to the inverse ratio of the numbers of teeth; hence, for helical gears, relating to the example shown in Figure 5.5, the following relations would be applied:

$$\omega_1 /\omega_2 = n_2/n_1 = D_2 \cos \psi_2 / D_1 \cos \psi_1 \tag{5.18}$$

where ψ is helix angle and other terms are as in spur gears. For helical gears with parallel shafts, the helix angles are the same, but when the shafts are not parallel, the relation between helix angles are:

$$\alpha = \psi_1 + \psi_2 \tag{5.19}$$

where α is the angle between the shafts, and the center distance relation is the same as for spur gears.

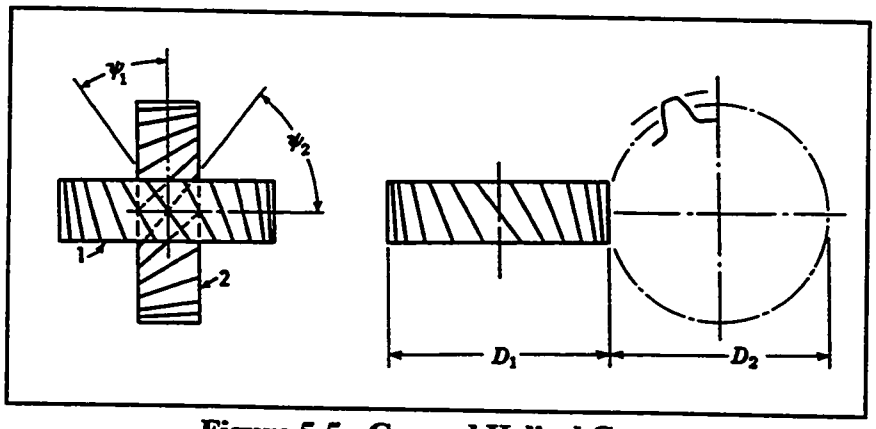


Figure 5-5 : Crossed Helical Gears

5.2.3.2 Bevel Gear

Intersecting shafts are connected by bevel gears. The pitch cones are analogous to the pitch cylinders of spur and helical gears. It can be proven that Lewis equation for bevel gears are as follows:

$$W = sFpy(1-F/L + F^2/(3L^2)) \approx sFpy(1-F/L) \qquad F/L << 1/3 \qquad (5.20)$$

Where **L** is pitch cone length. Note that a spur gear may be considered as limiting case of a bevel gear for which **L** is infinitely long or F/L = 0. The force **W** is not an actual force but an equivalent force acting at the large end of the tooth. **W** can be determined by dividing the torque on the gear by the radius R =D/2, where **D** is pitch diameter of the bevel gear.

Straight bevel gears, gears with intersected axes, are generally at right angles, and the element of the teeth are straight lines that converge at the apex of the pitch cone. In these kind of bevel gears, dynamic load and surface endurance will be dealt with as a spur gear. The face width in bevel gears, **F**, is equal to or greater than **10/p** and less than or equal to **L/3**.

As with a spur or helical gear, the pitch diameter of a bevel gear is related to its circular pitch and number of teeth by the formula:

$$D_g = N_g\, p/\pi.$$

The velocity ratio of bevel gears is the same as spur gears, but in intersect bevel gears, center distance can not be used in the calculation of the gears pitch diameters.

5.2.3.3 Worm Gear

This type of gear is extensively used to transmit power at high velocity ratios (300:1 or over) between nonintersecting shafts that are generally, but not necessarily at right angles. Since the teeth

of a worm gear are weaker than those of the worm threads, the design for strength may be based on the modified Lewis equation which was applied on the helical gears. The only difference will be on velocity factor, which in worm gears is as follow:

$C_v = 1200/(1200+V)$

Because of the sliding action between the worm and gear teeth, the dynamic forces are not so severe as in the regular forms of gears. In these kind of gears, because of energy losses in sliding friction, efficiency is important and defined as follow:

$$e = F_{gt}V_g/(F_{wt}V_w) \qquad (5.21)$$

$$= (\cos \phi_n - \mu \tan \lambda)/(\cos \phi_n + \mu \cot \lambda)$$

where e = efficiency of worm gear

F_{gt} = tangential force of the gear

V_g = tangential velocity of the gear

F_{wt} = tangential force of the worm

V_w = tangential velocity of the worm

ϕ_n = pressure angle in normal plane

μ = friction coefficient

λ = worm lead angle

There is no standardized method for determining the load limit for wear of worm gears. Manufacturers usually use one of the empirical methods backed by their experience with the particular type of worm and worm gear that they make.

The speed ratio of a worm gear is equal to the number of teeth on the worm wheel divided by the number of threads on the worm. Worm gears usually have at least 24 teeth, and the number

of gear teeth plus worm threads should be more than 40:

$$N_w + N_g > 40 \tag{5.22}$$

As with a spur or helical gear, the pitch diameter of a worm gear is related to its circular pitch and number of teeth by the formula:

$$D_g = N_g\, p/\pi.$$

The axial pitch, **p**, of the worm is equal to the circular pitch of the mating worm wheel. Its pitch diameter can be approximated by:

$$D_w = 2.4\, p + 1.1 \text{ (in.)} \tag{5.23}$$

Also, this pitch diameter is normally related to the shaft center distance as following:

$$C^{0.875}/3.0 \le D_w \le C^{0.875}/1.7 \tag{5.24}$$

In a single-threaded worm, the lead **L** and the axial pitch are equal; hence the lead of an n-threaded worm is equal to n times its axial pitch. The pitch lead angle **λ** is the angle between the tangent to the pitch helix and the plane of rotation. The pitch helix angle is the angle between the tangent to the pitch helix and an element of the pitch cylinder. As seen in Figure 5.6, following

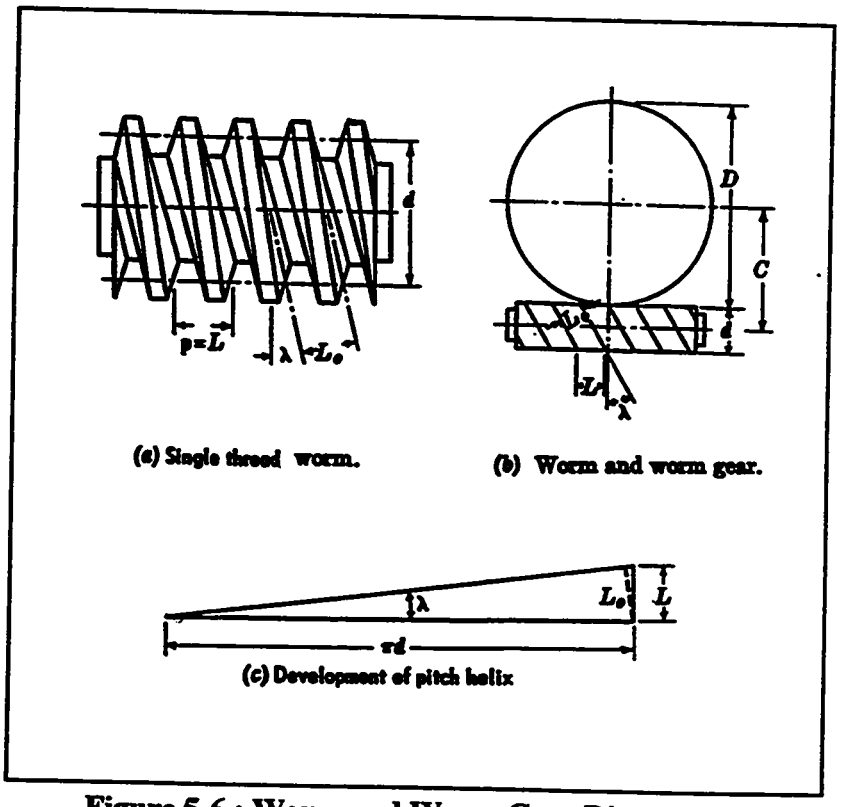


Figure 5-6 : Worm and Worm Gear Dimensions

relations apply to lead angle and center distance:

$$\lambda = L/\pi D_w \tag{5.25}$$

$$C = (D_w + D_g)/2 = L(\cot \lambda + r)/2\pi \tag{5.26}$$

The face width of the gear is as follow:

$$F \leq 0.5 D_{w,out} \tag{5.27}$$

where $D_{w,out}$ is the worm outside diameter.

5.2.4 Object-Oriented Approach in Gear Design

As can be observed, only simple design methods in different type of gears have been considered in the preceding sections. The stages of these methods must be organized in a way so that the inheritance and polymorphism features of the object-oriented approach can easily be applied on them. For example, during application of the Lewis equation in order to obtain strength of gear teeth, when considering dynamic effects due to the velocity of tooth action, in different types of gear, there are different relations referenced to calculate the velocity factor, C_v. In order to solve this diversity in the velocity factor calculation, one method (algorithm) is implemented in each type of gear class in order to calculate the velocity factor of that type of gear, and all of them are assigned a single name (i.e., A6). Then using polymorphism, by calling that method for a gear whose design process is in hand, the velocity factor of that gear will be calculated.

In some other stages of gear design, it is not possible easily to separate a part of the design stage and override it for a specific gear design. For instance, for obtaining the surface durability for general gear teeth, the Buckingham equation, $W_w = DFKQ$, is used, while in helical gears a modified equation of Buckingham, $W_w = DFKQ/\cos^2\psi$, is used. In this case, a new method for the Buckingham

equation can be written in a helical gear class which calls the Buckingham equation method in its superclass to obtain the value of W_w in general gear, then divides this value by $\cos^2\psi$ to calculate W_w for helical gear.

In other cases, these procedures alone might not be able to solve the problem. For example, in the superclass of the gears, the dynamic load equation which was applied on gear design is as follows:

$$W_d = W + (0.05V(FC + W) / (0.05V + \sqrt{(FC + W)})$$

While in helical gear the proper equation is as follows:

$$W_d = W + (0.05V(FC \cos^2 \psi + W) \cos \psi / (0.05V + \sqrt{(FC \cos^2 \psi + W)})$$

In this case, new parameters should be defined. For instance, as the first step, the dynamic load can be assumed as summation of two separate loads as follows:

$$W_d = W + W_i$$

where increment load, W_i, can be written as:

$$W_i = (0.05V(F_{eq} C + W) / (0.05V + \sqrt{(F_{eq} C + W)})$$

where F_{eq}, equivalent face width, is related to the face width of the helical gear but does not represent any geometrical quantity. In order to solve this case problem, a combination of previous procedures will be used. In the gear classes which have a helix angle (e.g., helical or worm gear), the method which calculates the equivalent face width will be overridden. While for the calculation of W_i for helical gear, the second procedure is used. That means that at first the W_i for a general gear is calculated by calling its superclass method, then multiplying its value by $\cos \psi$, then the W_i of the helical gear will be obtained.

There are still some cases which are not solved by the above procedures. For example, as mentioned, since the teeth of a worm gear are weaker than those of the worm threads, the design may be based on the application of the Lewis equation on worm gear teeth. In a case where the designer

wants to start the design of the worm teeth strength, there will be some problems to be solved. These problems occur because the terminology and parameters used in the worm design differ compared to a general gear. For example, there is an axial pitch in the worm instead of the circular pitch in general gear; or there is a pitch lead angle, λ, instead of the helix angle in helical gears.

There are two methods to solve this case problem. The first method is to define new related terms which are equivalent to their terms in general gear. For instance, the term ψ, helix angle for worm, is defined and related to previous term λ (i.e., $\psi = 90 - \lambda$). In this method, the worm is considered as a gear. In the second method, a worm can be assumed as a subclass of the power screw class, and its design process will be inherited from its superclass. This method is not considered in this research because the power screw class is not implemented.

5.2.5 Gear Force Analysis

The analysis of the forces on gears is very important when analyzing the connection of the gears and the shafts. In this section, the forces on different gears will be discussed.

5.2.5.1 Conventional Gear Force Analysis

The simplest gear pair is the spur gear. Forces between mating teeth of this type of gears are resolved at the pitch point into two components:

a) tangential component, F_t

b) radial component, F_r

where $F_r = F_t \tan \phi$ and $F_t = 33000\ hp/V$

V is pitch-line velocity in feet per minute, and $V = \pi D\ rpm/12$.

There is no axial force in spur gears.

The force components comparison on a spur and helical gear indicates that in helical gears the component $\mathbf{F_a}$ (axial force) is added. These components are determined as follows:

a) $F_t = 33000\ hp/V$ (5.28a)

b) $F_r = F_t \tan \phi$ (5.28b)

c) $F_a = F_t \tan \psi$ (5.28c)

and for bevel gears:

a) $F_t = 33000\ hp/V_{av}$ (5.29a)

b) $F_r = F_t \tan \phi \cos \gamma$ (5.29b)

c) $F_a = F_t \tan \phi \sin \gamma$ (5.29c)

where V_{av} (average pitch line velocity) $= \pi D_{av}$ rpm/12,

D_{av} (average pitch diameter) $= D - b \sin \gamma$,

b = tooth width,

γ = pitch cone length.

In worm and worm gears, for the usual 90 deg. shaft angle, there are tangential, axial, and radial force components acting on a worm and a worm gear. The worm tangential force is equal to the gear axial force and vice versa ($F_{wt} = F_{ga}$, and $F_{gt} = F_{wa}$). Also, the worm and the gear radial or separating forces are equal ($F_{wr} = F_{gr}$).

$F_{gt} = F_{wa} = F_n \cos \phi_n \cos \lambda - \mu F_n \sin \lambda$ (5.30a)

$F_{wt} = F_{ga} = F_n \cos \phi_n \sin \lambda + \mu F_n \cos \lambda$ (5.30b)

$F_{gr} = F_{wr} = F_n \sin \phi_n$ (5.30c)

where $\mathbf{F_n}$ is force on the worm and gear in normal plane.

All these force analyses are for the simple case of each gear type. For example, parallel shafts for helical gears, and right angle shafts for worm gears. In the following section, a more general version will be examined.

5.2.5.2 Generalized Gear Force Analysis

In rotational components (e.g. shafts, gears, etc.) axes are of high importance. This is due to the fact that dimensions are parallel or perpendicular to the axis. Thus, it would simplify the component's parametric calculations if the coordinate system of these components were considered in a way that one of the coordinate axes coincided with the component axis. In this research, the rotational axis of these components, will be coincided with their x-axis. This method also improves the encapsulation of the components parametric calculations which coincide with the object-oriented objectives.

In order to simplify the relation between components coordinate systems and the geartrain coordinate system, it can be assumed that the shaft axes lies on any horizontal plane (parallel to x-z plane) of the geartrain coordinate system. Most of the geartrains design fall in this category. A more general, but yet simple coordinate system, is considering the axes of two shafts connected by gears to pass through parallel planes (Figure 5.7). In this case we process each

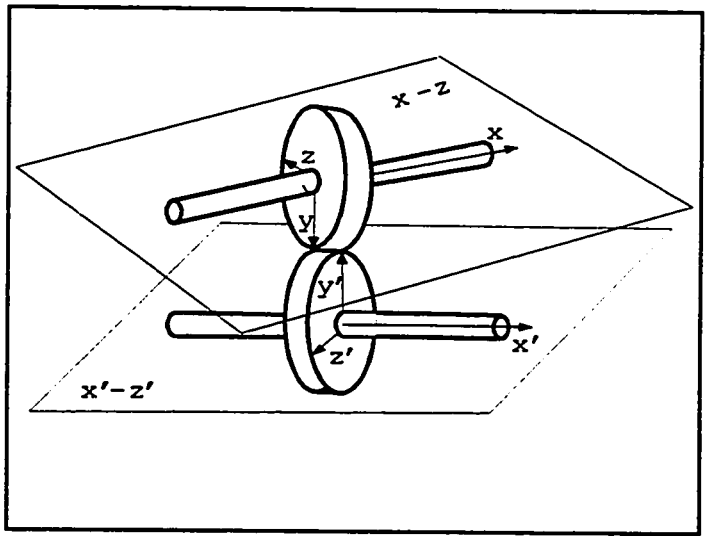


Figure 5.7 : Gear to Gear Connection

connection at a time, so it's not necessary that all planes to be parallel. Two other coordinate axes 'y' and 'z' will be perpendicular to x-axis. These axes will be defined according to their connection which will be discussed later in this chapter.

This assumption simplifies the design implementation without losing any generality. In order to find these parallel planes, a simple connection of the center of gears, will create a line (oo') perpendicular to both shafts axes. The two parallel planes which pass through the two shafts axes and are perpendicular to line oo' are those mentioned planes. These planes may not be horizontal or parallel to the original planes of the geartrain coordinate system. Also one single shaft axis may lie in more than one plane when considering more than one connections with the single shaft. The y-axes in this component coordinate system will be in oo' direction, and the z-axis will be perpendicular to the 'x' and 'y' axes in a way that makes a right-handed coordinate system. The 'x' and 'z' axes, make the component plane. In gears with intersected axes (e.g. bevel gears) where it is impossible to have two parallel planes, there will be only one plane which contains both shafts' rotational axes. In this case, the y-axis is the connection of the gear center and its contacting point, so the 'y' and 'x' axes of these gears will lie on the plane while the z-axis will be perpendicular to the plane.

In the beginning of the design process, the input shaft is the only component whose position with respect to the geartrain coordinate system is known. Other components positions will be determined by calculating related parameters values during the design process. All the dimensions of the input shaft (e.g. the diameter and length of the shaft) might not be known. Hence, the first step will be the calculation of the diameter and the length of the input shaft. The simplest shaft design is a shaft with one gear and two bearings mounted on it. In order to calculate the diameter and the

length of this shaft one needs to know all the forces applied on it from all mounted components. Before proceeding the shaft design, the gear load on the shaft must be calculated.

There are different type of gears to be considered. Efficiently using the object oriented system, each type will need separate consideration. A general type for all gears (class Gear) will be defined, and any specific gear will be its subclasses. Any common characteristic among all types of gears will be implemented in the abstract class of Gear. Meshed gears' type relates to the orientation of gears axes with respect to each other, their transmission power and rpm. Meshed gears are divided into three different categories according to the orientation of their rotational axes: the parallel, nonparallel and intersected axes categories. This project will examine four different type of gears, bevel, helical, spur and worm gears. Spur and parallel helical gears will fall in the first category (parallel axes), crossed helical and worm gears in the second category (nonparallel axes), and bevel gears in the third category (intersected axes).

Another method to generalize gear types is to define pair gears as a class of objects. There are some problems applying this method. Some of these difficulties are as follows:

1- There may be more than one gear meshed with a specific gear.

2- Different orientation of axes in different pair of gears need to reference different subclasses for each type of gear. In order to solve this problem the same method will be used as when there were no pair gears, so there is no need for a pair gears object.

3- There are so many parameters for each gear (e.g. diameter, speed, number of teeth, ...) . Hence, two separate sets of instance variables for each class of pair gears should be considered.

4- If the materials of the pair gears are different, there will be different relations for the contact point and the strain and stress calculation. With these problems, in this research, the first method is

recommended.

In the mechanical design, connection means exchanging force and torque between connected components. In three dimensional design, both the force and the torque have three components. The three force components are named R_x, R_y and R_z, and torque components are named M_x, M_y, M_z. Considering only one piece of gear in general (Figure 5.8), there will be a force F applying on the tooth of the gear which has an angle of ϕ (pressure angle in the rotational plane y-z) with the z-axis and an angle of φ (helix angle in the x-z plane) with the z-axis. The acting point of the force F is assumed to be on the y-axis, so we have:

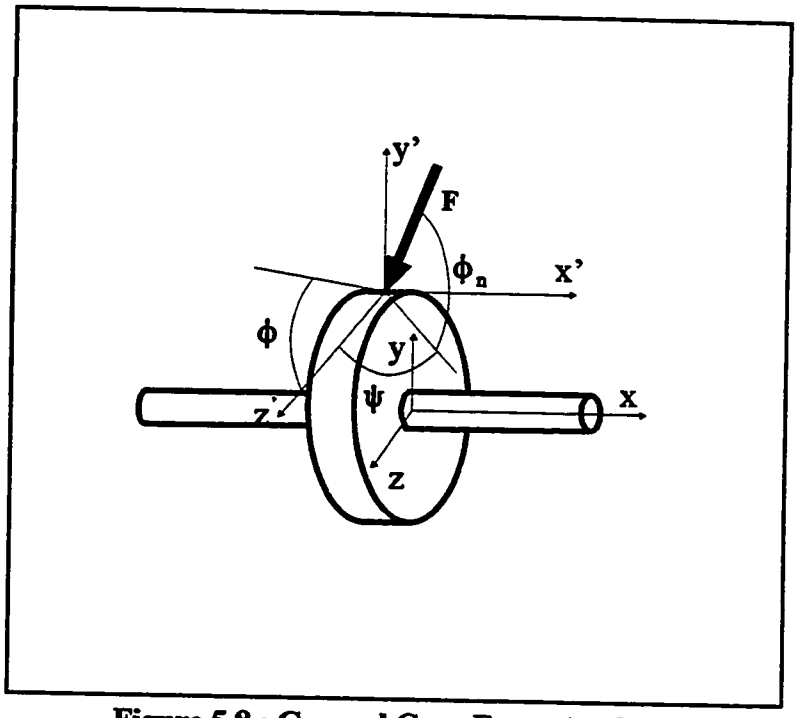


Figure 5.8 : General Gear Force Analysis

$$\begin{cases} F_a = R_x = F\cos\phi_n \sin\psi \\ F_r = R_y = F\sin\phi_n \\ F_t = R_z = F\cos\phi_n \cos\psi \end{cases} \quad and \quad \begin{cases} M_x = T \\ M_y = 0 \\ M_z = -R_x D/2 \end{cases} \tag{5.31}$$

where ϕ_n is pressure angle in normal plane, $\tan\phi_n = \tan\phi \cos\psi$, R_x is the trust, R_y is the separating force, R_z is the tangential force, M_x is the torque and M_y and M_z are the bending moment on the gear.

The value of the T (torque) is usually known and the equations can be rewritten: $R_z = 2T/D$

or:

$F = 2T/(D \cos\phi_n \cos\psi)$. Hence:

$$\begin{cases} R_x = 2T\tan\psi/(D\cos\phi_n) \\ R_y = -(2T/D)\tan\phi_n \\ R_z = 2T/D \end{cases} \quad and \quad \begin{cases} M_x = T \\ M_y = 0 \\ M_z = -R_x D/2 \end{cases} \tag{5.32}$$

These simplified (but generic) equations can apply to all type of gears. It is only important to know that, in the spur gear, φ is zero and also, in the worm, instead of the helix angle (ψ) a lead angle (λ) exists. Where λ is an angle in the x-z plane with respect to the x-axis. Hence, in a worm, it is assumed $\psi = 90^\circ - \lambda$.

Therefore all force and torque components applying on the teeth of any gears can be calculated, knowing **T** (transmission torque), **D** (pitch diameter of gear), ϕ (pressure angle) and ψ (helix angle or 90° - lead angle 'λ' in worm). A relation between the forces and the torques of these components and the counter-components of the gear mate is also needed. Hence, its necessary to know the orientation of the gear mate with respect to the gear. In the simplest case when rotational axes of gears are parallel (first category), two planes as in Figure 5.7 will be considered as x-y plane. The z-axes of both gears will be perpendicular to this plane. This case includes spur and parallel helical gears. The relations between the two counter component forces and torques are as follows:

$$\begin{cases} R_{x,t} = -R_{x,h} \\ R_{y,t} = R_{y,h} \\ R_{z,t} = R_{x,h} \end{cases} \quad and \quad \begin{cases} M_{x,t} = (D_t/D_h)M_{x,h} \\ M_{y,t} = M_{y,h} = 0 \\ M_{z,t} = -(D_t/D_h)M_{z,h} \end{cases} \tag{5.33}$$

where subscript **t** stands for the tail and subscript **h** for the head component of the connection object.

In non-parallel axes gears (second category, Figure 5.9), crossed helical and worm gears are included. The relation between the two meshed gears (the head and the tail in the connection object) will be as follows:

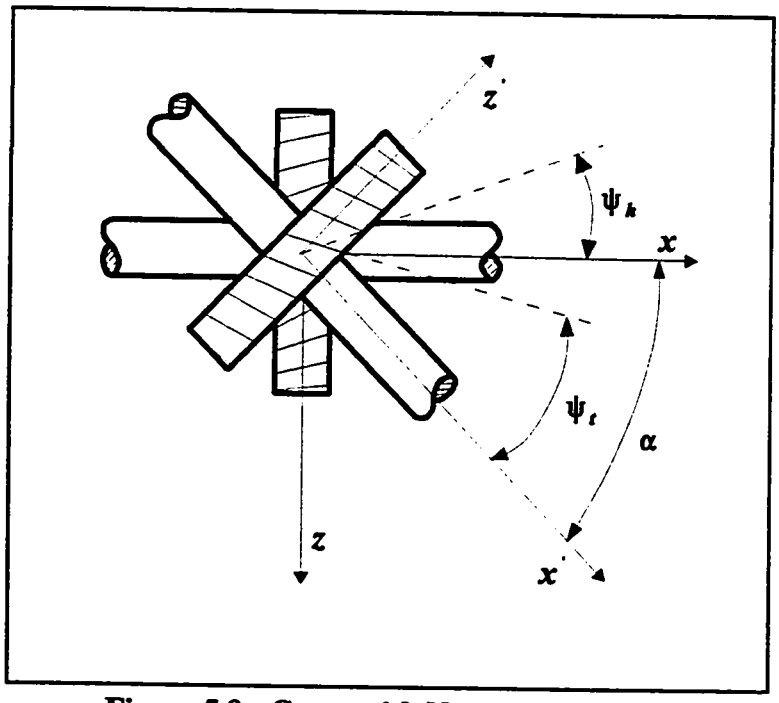


Figure 5.9 : Gears with Non-parallel Axes

$$\begin{cases} R_{x,t} = -R_{x,h}\cos\alpha - R_{z,h}\sin\alpha \\ R_{y,t} = R_{y,h} \\ R_{z,t} = R_{z,h}\cos\alpha - R_{x,h}\sin\alpha \end{cases} \quad or \quad \begin{cases} R_{x,h} = -R_{x,t}\cos\alpha - R_{z,t}\sin\alpha \\ R_{y,h} = R_{y,t} \\ R_{z,h} = R_{z,t}\cos\alpha - R_{x,t}\sin\alpha \end{cases} \qquad (5.34)$$

here $\alpha = \psi_t + \psi_h$, ψ_t and ψ_h are helix angles of the tail and the head gears. When $\alpha = 0$, that means parallel axes, these relations will be the same as previous ones.

In third category (intersected axes gears, Figure 5.10), considering bevel gears, the relation between the two meshed gears (the head and the tail in the connection object) will be as follows:

$$\begin{cases} R_{x,t} = -R_{x,h}\cos\beta + R_{y,h}\sin\beta \\ R_{y,t} = R_{y,h}\cos\beta + R_{x,h}\sin\beta \\ R_{z,t} = R_{z,h} \end{cases} \quad or \quad \begin{cases} R_{x,h} = -R_{x,t}\cos\beta + R_{y,t}\sin\beta \\ R_{y,h} = R_{y,t}\cos\beta + R_{x,t}\sin\beta \\ R_{z,h} = R_{z,t} \end{cases} \tag{5.35}$$

where $\beta = \gamma_t + \gamma_h$, γ_t and γ_h are pitch cone angles of the tail and the head bevel gears.

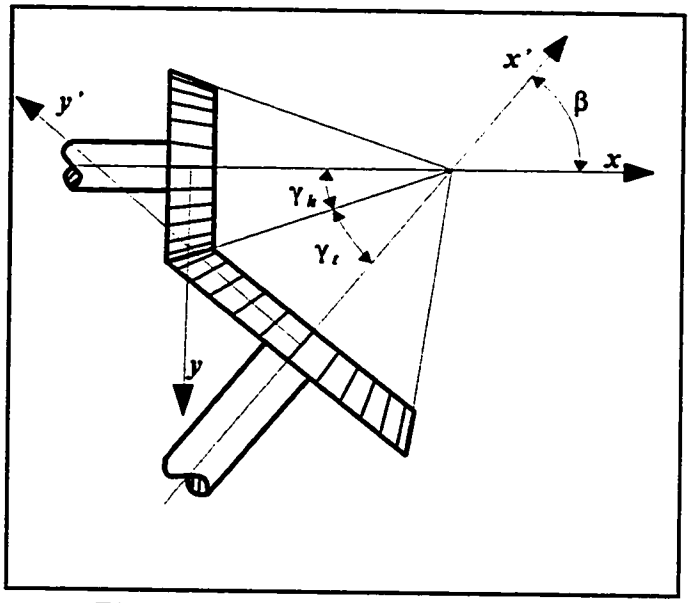


Figure 5.10 : Intersected Axes Gears

Although in this assumption of categories one of these angles (α and β) or both of them are zero. To solve this nonconsistency, both angles in a general gear are to be considered. This indicates the existence of non-parallel and non-intersecting gears (skew bevel gears). The detail design of this type of gears will not be included in this research, but by considering it as an abstract gear the problem will be solved. It is assumed α angle to be in the x-y plane of the tail component with respect to the x-axis, and β angle, in the x-z plane of the tail component, with respect to the x-axis. This simplifying assumption will not lose any generalities. Hence, the relations of the tail and the

head gear forces will be as follows:

$$\begin{cases} R_{x,t} = -R_{x,h}\cos\alpha\cos\beta + R_{y,h}\sin\beta - R_{z,h}\sin\alpha \\ R_{y,t} = +R_{x,h}\cos\alpha\sin\beta + R_{y,h}\cos\beta \\ R_{z,t} = -R_{x,h}\sin\alpha\cos\beta + R_{z,h}\cos\alpha \end{cases} \tag{5.36}$$

These are generic equations for the calculation of any type of gear forces according to its mate gear. Hence the moment equations of the mate gears will be as follows:

$$\begin{cases} M_{x,t} = M_{x,h}\left(\dfrac{D_t\cos\psi_t}{D_h\cos\psi_h}\right) \\ M_{y,t} = M_{y,h} = 0 \\ M_{z,t} = -\dfrac{D_t}{D_h}\left(M_{z,h}\cos\alpha\cos\beta - D_h R_{y,h}\sin\beta + M_{x,h}\sin\alpha\right) \end{cases} \tag{5.37}$$

In these relations, it is assumed that there is no loss of energy in the transmission power, meaning the efficiency of the gears are %100.

5.3 Shaft Design Generalization

As mentioned, the simplest shaft for design is a shaft with one gear and two bearings mounted on it (Figure 5.11). A determinate problem is to assume that the first bearing (B1) tolerates both thrust and radial forces, but the second bearing (B2) tolerates only the radial force. The loads

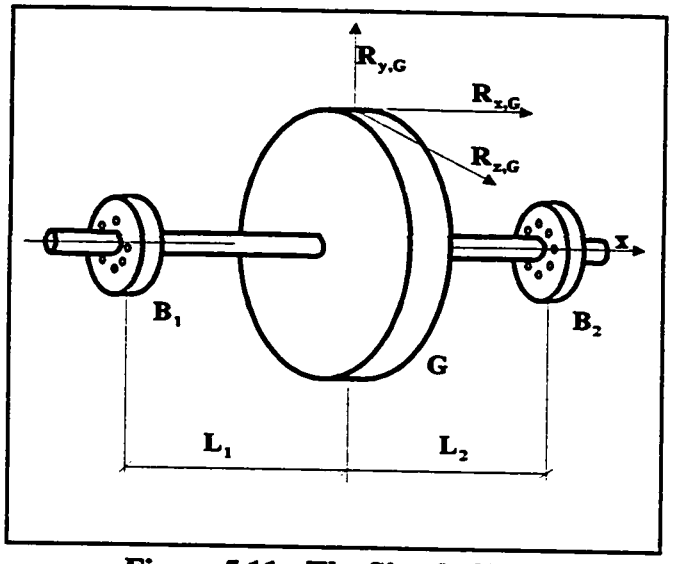


Figure 5.11 : The Simple Shaft

on these bearings will be as follows:

$$\begin{cases} \Sigma F_x = 0 \quad \Rightarrow R_{x,B1} = -R_{x,G} \\ \Sigma M_{z,B_2} = 0 \Rightarrow R_{y,B1} = -(D_G R_{x,G}/2 + L_2 R_{y,G})/(L_1+L_2) \\ \Sigma M_{y,B_2} = 0 \Rightarrow R_{z,B1} = -L_2 R_{z,G}/(L_1+L_2) \end{cases} \tag{5.38}$$

and

$$\begin{cases} \Sigma M_{z,B_1} = 0 \Rightarrow R_{y,B2} = (D_G R_{x,G}/2 - L_1 R_{y,G})/(L_1+L_2) \\ \Sigma M_{y,B_1} = 0 \Rightarrow R_{z,B2} = -L_1 R_{z,G}/(L_1+L_2) \end{cases} \tag{5.39}$$

here, it is assumed that the contact point is on the x-y plane. If there are more than one contact points on the gear or there are more than one mounted gear on the shaft, all contact points can not be on the x-y plane. Hence, reactions on the bearings at the both ends of the shaft will be as follows:

$$
\begin{cases}
R_{x,B1} = -\sum_{i=1}^{n} R_{x,Gi} \\
R_{y,B1} = -\dfrac{\sum_{m=n}^{1}\left[(R_{ym}\cos\delta_m - R_{zm}\sin\delta_m)\times\left(\dfrac{F_m}{2}+L_m+\sum_{i=m+1}^{n}(F_i+L_{i+1})+\dfrac{F_{B2}}{2}\right)+(R_{xm}\cos\delta_m)\times\dfrac{D_m}{2}\right]}{\dfrac{F_{B1}}{2}+\sum_{i=1}^{n}(L_i+F_i)+(L_{n+1}+\dfrac{F_{B2}}{2})} \\
R_{z,B1} = -\dfrac{\sum_{m=n}^{1}\left[(R_{ym}\sin\delta_m + R_{zm}\cos\delta_m)\times\left(\dfrac{F_m}{2}+L_m+\sum_{i=m+1}^{n}(F_i+L_{i+1})+\dfrac{F_{B2}}{2}\right)+(R_{xm}\sin\delta_m)\times\dfrac{D_m}{2}\right]}{\dfrac{F_{B1}}{2}+\sum_{i=1}^{n}(L_i+F_i)+(L_{n+1}+\dfrac{F_{B2}}{2})}
\end{cases}
\tag{5.40}
$$

$$
\begin{cases}
R_{y,B2} = -\dfrac{\sum_{m=1}^{n}\left[(R_{ym}\cos\delta_m - R_{zm}\sin\delta_m)\times\left(\dfrac{F_m}{2}+L_m+\sum_{i=1}^{m-1}(F_i+L_{i+1})+\dfrac{F_{B1}}{2}\right)-(R_{xm}\cos\delta_m)\times\dfrac{D_m}{2}\right]}{\dfrac{F_{B1}}{2}+\sum_{i=1}^{n}(L_i+F_i)+(L_{n+1}+\dfrac{F_{B2}}{2})} \\
R_{z,B2} = -\dfrac{\sum_{m=1}^{n}\left[(R_{ym}\sin\delta_m + R_{zm}\cos\delta_m)\times\left(\dfrac{F_m}{2}+L_m+\sum_{i=1}^{m-1}(F_i+L_{i+1})+\dfrac{F_{B1}}{2}\right)-(R_{xm}\sin\delta_m)\times\dfrac{D_m}{2}\right]}{\dfrac{F_{B1}}{2}+\sum_{i=1}^{n}(L_i+F_i)+(L_{n+1}+\dfrac{F_{B2}}{2})}
\end{cases}
\tag{5.41}
$$

where **n** is the number of gear connections of mounted gears. For example if there are two mounted gears on a shaft, with one and two connections, then **n**= 1+2=3. '**m**' is subscript for component number **m** respectively from shaft left hand side. δ_m is the angle between the radius passing from

gear connection and shaft's 'y' axis, and $\mathbf{F_m}$ is the face width of the component **m**. $\mathbf{L_i}$ is the shaft partial length between mounted component number **I** and **I+1**.

There are two problems in the calculation of bearings reaction from these relations. First, to calculate the summation series in these relations, it is necessary to know the order of mounted components in advance. Second, in these relations, it is assumed that bearings are in both ends of the shaft. This may not always be true. By assuming that bearings can be in any position on shaft, and knowing the order of components mounted on the shaft, the following relations is result:

$$\begin{cases} \Sigma M_{z,l} = 0 \Rightarrow a_{11}R_{y,Bl} + a_{12}R_{y,B2} = c_{zl} = R_{ym} \times \sum_{i=1}^{m} L_i + \sum_{i=1}^{m} M_{zm} \\ \Sigma M_{z,r} = 0 \Rightarrow a_{21}R_{y,Bl} + a_{22}R_{y,B2} = c_{z2} = R_{ym} \times \sum_{i=m}^{n} L_i + \sum_{i=m}^{n} M_{zm} \end{cases} \tag{5.42}$$

and

$$\begin{cases} \Sigma M_{y,l} = 0 \Rightarrow a_{11}R_{z,Bl} + a_{12}R_{z,B2} = c_{yl} = R_{zm} \times \sum_{i=1}^{m} L_i + \sum_{i=1}^{m} M_{ym} \\ \Sigma M_{y,r} = 0 \Rightarrow a_{21}R_{z,Bl} + a_{22}R_{z,B2} = c_{y2} = R_{zm} \times \sum_{i=m}^{n} L_i + \sum_{i=m}^{n} M_{ym} \end{cases} \tag{5.43}$$

where $\mathbf{a_{11}}$ = distance of the bearing $\mathbf{B_1}$ center from the shaft left end,

$\mathbf{a_{12}}$ = distance of the bearing $\mathbf{B_2}$ center from the shaft left end,

$\mathbf{a_{21}}$ = distance of the bearing $\mathbf{B_1}$ center from the shaft right end,

$\mathbf{a_{22}}$ = distance of the bearing $\mathbf{B_2}$ center from the shaft right end.

$M_{y,l}$,$M_{y,r}$,$M_{z,l}$,and $M_{z,r}$ are moments in left and right handside of the shaft around 'y' and 'z' axes.

Reactions of the bearings ($R_{y,B1}$, $R_{z,B1}$, $R_{y,B2}$ and $R_{z,B2}$) can be calculated from these equations, and for R_x the same relation will result as before. In some designs each bearing tolerates only one direction of R_x, for example, negative R_x by bearing one and positive R_x by bearing two.

5.3.1 Shaft Minimum Partial Length

For calculation of the minimum partial length of the shaft for each mounted component, it is required to know the width of the component and the distances between this component and its neighbors. The face width of the gears the (F_i) and the bearings (F_{Bi}) will be calculated in their PDDs, but for bearings, because the load value is needed to calculate the face width, an initial assumption value will be given. For the distances between mounted components (L_i) there are some minima but the user can increase them as required. In the first category when, the axes of gears are parallel (e.g. spur gears or parallel helical gears), the minimum partial length of the shaft will be at least the width of the gears

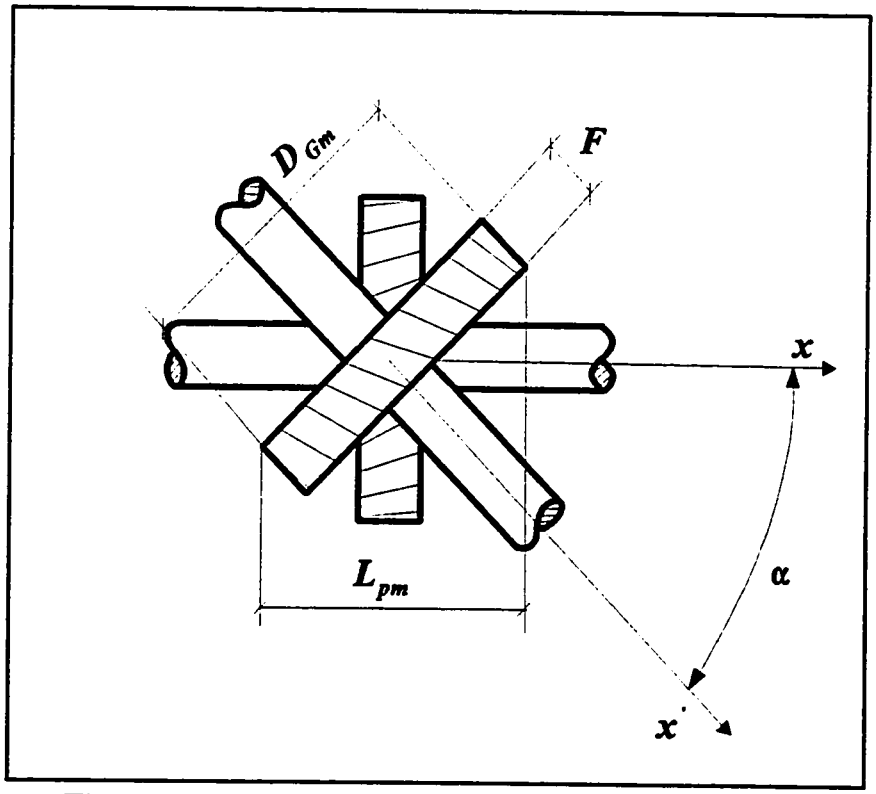


Figure 5.12 : Shaft Minimum Partial Length for Gears

(F). In the second category, when axes are not parallel (e.g. non-parallel helical or worm gears), this minimum depends on axes' angles (Figure 5.12) as follows:

$$L_{pm} = D_{Gm}\sin\alpha + F\cos\alpha \tag{5.44}$$

where L_{pm} is the minimum partial length of the shaft and **F** is the gear face width and $\mathbf{D_{Gm}}$ is the mate-gear diameter. As it is seen, when α is zero (parallel gears), $L_{pm} = F$ and when $\alpha = 90^{\circ}$ (worm gears),

$L_{pm} = D_{Gm}$. In the third category (bevel gears), the modified formula of L_{pm} which is a more general form, will be used:

$$L_{pm} = D_{Gm}\sin(\alpha+\beta) + F\cos\gamma_{Gm}\cos\alpha \tag{5.45}$$

where it is assumed that α or β or both are zero, and γ_{Gm} is bevel mate-gear cone angle which is zero for non-bevel gears. There is an exception for the caculation of L_{pm} if the bevel gear is at the end of shaft (Figure 5.13), or the diameter of components after the bevel gear are smaller than the mate-gear diameter, where $L_{pm} = F$.

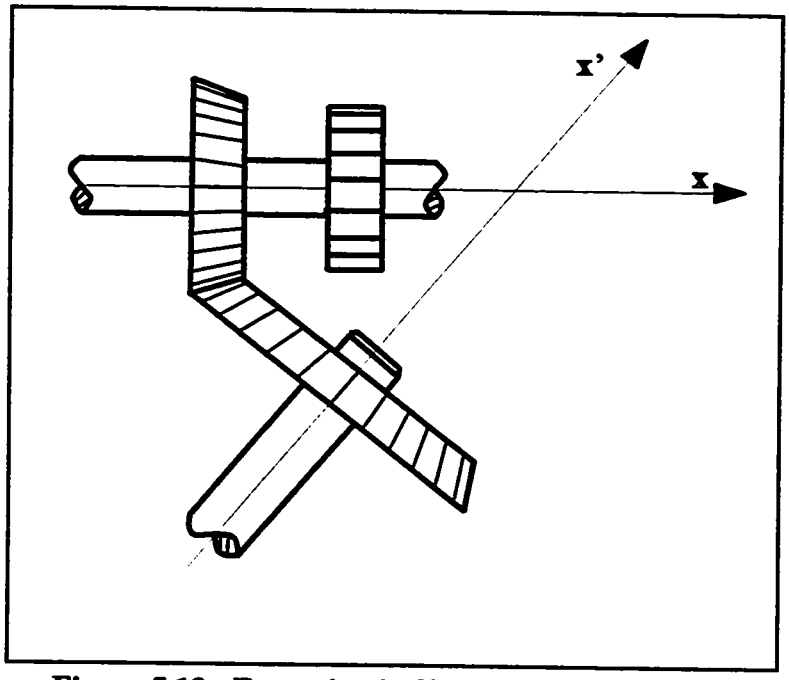


Figure 5.13 : Exception in Shaft Minimum Partial Length

In all the cases for the minimum partial length of the shaft, this minimum will be multiplied by a coefficient to cover the

gear's hub and the distance between the components.

5.3.2 Shaft Design

The action of the loads on a shaft is generally combinations of torsion, bending and axial loads. The stress created by each load can be calculated as follows:

$$\tau = \frac{T \times r}{J}$$

where τ *is shearing stress,* T *is torque,* r *is shaft radius and* J *is polar moment of inertia*

$$s_b = \frac{M \times r}{I}$$

where s_b *is bending load stress,* M *is bending moment and* I *is moment of inertia*

$$s_a = \frac{R_x}{A}$$

where s_a *is axial load stress,* R_x *is axial force and* A *is cross-sectional area*

In most design books, a combination of torsion and bending load is considered for shaft design. For a solid circular shaft the following relations are noted: $J = \pi D_{sh}^3/16$, $I = \pi D_{sh}^3/32$ and $A = \pi D_{sh}^3/4$ (where D_{sh} is the shaft diameter). The design of the shaft is based on the maximum-shear-stress theory; hence it is necessary to determine the maximum combined shear stress in the shaft due to the applied torsion and bending moments. The maximum combined shear stress is as follow:

$$s_{max} = \sqrt{\left(\frac{s_b}{2}\right)^2 + \tau^2} \tag{5.46}$$

and substituting s_b and τ it results:

$$s_{max} = \frac{16}{\pi D_{sh}^3}\sqrt{T^2 + M^2} \tag{5.47}$$

where T is M_z and $M^2 = (M_x)^2 + (M_y)^2$.

If assumed that the whole shaft length has the same diameter (D_{sh}), the simplest method to calculate is from following formula:

$$D_{sh} = \sqrt[3]{\frac{16\sqrt{T^2 + M^2}}{\pi S_{all}}} \tag{5.48}$$

where S_{max} substituted by S_{all} the allowable stress.

In a real shaft's design usually there are diameter changes which causes steps on the shaft. In a stepped shaft, stress concentration factor (K_t) is of high importance. This factor depends on D_{sh}/d_{sh} and r/d_{sh} where D_{sh} and d_{sh} are the shaft diameters at both side of the step and r is the fillet radius. These relations have been given in different design books as graphs for different values of those parameters. Assuming a combination factor for all loads D_{sh} can be calculated as follows:

$$D_{sh} = \sqrt[3]{\frac{16}{\pi} \cdot \frac{K_t \sqrt{T^2 + M^2}}{S_{all}}} \qquad (5.49)$$

As seen from this formula D_{sh} and K_t are related to each other. So for calculation of this factor it is necessary to know the shaft diameter itself. Hence, for the calculation of the initial value of diameter, it is assumed K_t as unit. Then, when having a value for the diameter, K_t can be obtained according to that value. This step can be repeated until an acceptable value for the diameter of the shaft is founded in that region.

Prime factors in calculating a shaft diameter are torque ($T=M_z$) and bending moments (M_x and M_y), but in some shafts (e.g. shafts with bevel or worm gears) the effect of the axial force (R_x) is comparable with the effect of the bending moments. So, the normal stress will be a combination of the stresses caused by both the axial force and bending moments. Having considered different kind of loads, it will be difficult to obtain a combined concentrated factor for all loads, and also it will be more generic to calculate the real stress of each load individually by considering the concentrated factor of that type of load. For doing this, different stress can be calculated as follows:

$$S_{R_x} = K_{t1}\frac{R_x}{A} \qquad S_{M_x} = K_{t2}\frac{(d/2)M_x}{J}$$
$$S_{M_y} = K_{t3}\frac{(d/2)M_y}{I} \qquad S_{M_z} = K_{t3}\frac{(d/2)M_z}{I} \qquad (5.50)$$

where K_{t1} is axial load concentration factor, K_{t2} is torque concentration factor and K_{t3} is bending concentration factor. Hence normal and shear stress can be calculated as follows:

$$\sigma = \sqrt{S_{R_x}^{\,2} + S_{M_y}^{\,2} + S_{M_z}^{\,2}} \quad and \quad \tau = S_{M_x}$$

$$then \quad \sqrt{\left(\frac{\sigma}{2}\right)^2 + \tau^2} = S_{max} < S_{all} \tag{5.51}$$

In the case when there is a keyway on the shaft (Figure 5.14), the inner diameter in the shaft design calculations will be considered. It is also possible to check the maximum shear stress in the section A-A which is in the risk of failure.

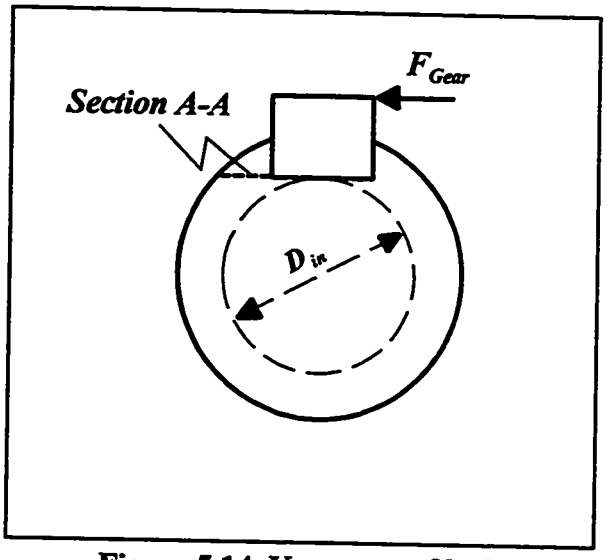


Figure 5.14 Keyway on Shaft

Chapter 6

CAID Environment

6.1 Introduction

In this chapter the identification of the classes will be considered. Instances of these classes, or objects, can be physical or functional ones. The first step is to study the design problem domain. Then simply list the names of significant classes and objects, using meaningful names that imply their semantics. Such a list defines a common vocabulary for better communication among different parts of the system and individuals who are working with the system.

By physical decomposition of the system, the relationship among these elements can be shown in an appropriate diagram. This idea will be used to derive the Objects Relations Diagram. Having determined the relationship among the objects, the external behavior of each object can be defined easily. In this step the boundaries of the abstractions important to the design can be defined.

The second step involves the collection of similar objects as a class and arranging the classes in a hierarchy using the inheritance mechanism. Implementation of these classes will complete the class hierarchy. By identifying this class hierarchy, the understanding of the design elements will be greatly simplified. Implementation of important components of the CAID system such as the design engine, the design model and the part design diagram will be elaborated on in individual sections of this chapter.

6.2 Objects and Classes Identification

The identification of classes and objects is the fundamental stage of the object-oriented design process. Identification involves discovery, invention, insight and experience. At this stage, the significant classes and objects have to be discovered, then important mechanisms that provide the behavior required from the objects have to be invented. By studying the requirements of the problem, the vocabulary of the problem domain can be learned. This vocabulary is a good source for choosing the design classes and objects. The candidate classes and objects of the design will consists of the tangible things in the problem domain, their responsibilities, and the events that may occur.

At the beginning, only the boundaries of the abstractions which are important to the design will be defined. Hence, at this point of the design process, the chosen classes and objects are candidates only. Some minor changes are expected in these boundaries due to the design progress. That is because more relations among these classes and objects are discovered over time as the design of the mechanisms which have been invented improves. In spite of the minor changes of their outside views, the classes and objects which are identified at this early stage of the design are present through the entire design process. This is only one of the consequences of the object-oriented systems, since the objects in this kind of a system reflect the model of the real world.

The results of this step may vary from informal to formal. At the beginning of the design process, listing the names of important classes and objects will be sufficient, and meaningful names that represent their entities are very useful. During the design process some new objects can be added to this list. This list consists of some classes, some objects, and attributes of objects. At the end of the design process, the meaning of these abstractions and mechanisms will be formally specified by

implementing the appropriate classes and objects.

This step takes a small amount of time for the designer with object-oriented design experience. For example, by studying the simple gear train design problem of Figure 6.1, the most significant tangible elements like shafts, gears, keys, bearings and housings can be considered. The connection of these elements to each other are also important for the design. There are also other factors like horsepower, material of elements and so on, which are the attributes of elements that have to be considered.

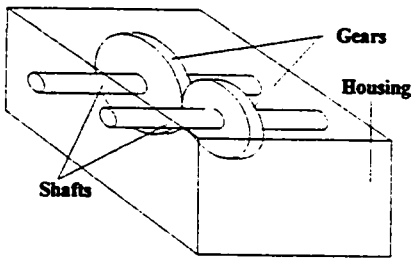


Figure 6.1 : A Gear Train and Its Elements

By studying different gears or different connections one can find similar behavior among similar objects. For instance, different gears (i.e., spur, bevel, helical and worm gear) behave like each other. Hence, one can define a generic class called 'Gear' to share all common behavior of all gears, then each specific gear will be defined as a Gear subclass. These subclasses will inherit the common behavior from the Gear class, and their different behavior will be implemented in their own classes. The same will be applied to the Connection class and its subclasses.

6.3 Objects External Behavior

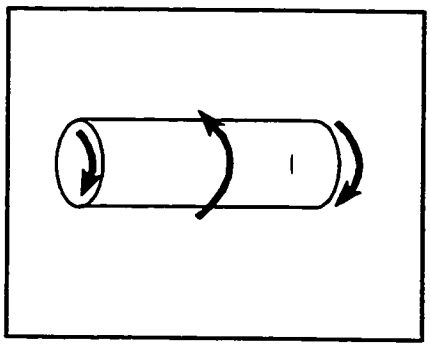
Establishing the meanings of the classes and objects identified from the previous step is the main task in this step. Here, each class will be viewed from the perspective of its interface, i.e.,the exterior. Hence, the functions (tasks) that those objects might undergo for each other will be identified. Writing a script for each object is one of the useful techniques to guide this process. This script should include the characteristic behaviors of the object during its lifetime.

This step is much harder than the first step, but having determined the relationships among the objects in the previous step will facilitate defining the external behavior of each object. Finding classes and objects is the easy part; deciding upon the protocol of each object is hard. The process of object-oriented design becomes iterative at this point. Implementing the external behavior for a given object may require some change in the meaning of another object. This only shifts the boundaries of the system key abstractions, but it does not affect their existence.

Object-oriented design has an incremental nature, and these steps reflect its very nature. The templates drafted in the previous step will be refined in this step. This will help to document the decisions regarding the meaning of each class and object. At this point of the system implementation, all of the static and dynamic semantics of each key abstraction and mechanism should be documented. Documenting any new mechanism which might have been invented in this step of the process, it will be possibly necessary to draft a new objects relations diagram. The system can be analyzed and evaluated at the end of this step.

Machine elements such as shafts, gears and the like are the basic components which are used repeatedly in building machines. The structure and behavior of each of these elements are fairly well established and can be characterized by a known set of parameters. For example, in the case of a gear

train, a shaft usually has a cylindrical structure and its behavior can be describe as a torsion due to a load applied to it. The main parameters for this case are: diameter, length, material properties and so on (Figure 6.2). In general, these parameters can be grouped under heading of geometry, material, load, and so on.



```
Shaft class methods:
        new
        Parameters
Shaft methods:
        addToBearingSet: aBearing
        addToGearSet: aGear
        bearingList
        diameter
        diameter: aDiameter
        gearList
```

```
MechElement subclass: #Shaft
  instanceVariableNames:
    'diameter length material speed torque
     maxLoad type bearingSet gearSet'
```

Figure 6.2 : A Shaft's External Behavior and Its Implementation in Smalltalk

6.4 Classes Hierarchy

One of the important steps in using object-oriented programming in the design domain is to drive the hierarchy of the system's elements. As mentioned, the system in its design process is broken down into small easily understood and described components. Each system can be categorized in different levels. The most basic level is the Element. An Element is a fundamental mechanical building block, representable physically as a single part, such as a gear or a shaft.

Element can be designed as a main class, and different design domain elements (e.g.

Electrical or Mechanical Elements) can be defined as subclasses of the Element class (Figure 6.3). For organizing each design system's elements in subclasses some criteria is necessary. For Mechanical Elements the following criteria can be assumed:

A Mechanical Element can be classified by...

1-... application (in the design domain).

- In this category, some elements act as a fastener (bolts, nuts, ...), some as a transmitter (gears, shafts, ...), and so on.

2-... geometric shape.

- Here the shape of the elements is important and the designer wants to know what each element looks like (e.g. cylindrical, cubic, ...)

3-... according to simple or compound parts.

- When a component of the design is studied, it can be as simple as a shaft, or as complicated as an automobile engine. As mentioned, a design can be broken down into components. This decomposition can be recursively continued until all the components become simple elements where they can not be decomposed anymore. Looking at a machine in each different level of its decomposition, the components in each level can be simple or compound. In some literature they are called atoms and molecules [1].

4-... according to solidity (solid, fluid, flexible...)

- In this category, solidity of the element is important, because physical laws applied upon them are totally different. For example, analysis of a solid element in a machine is completely different from a fluid element in a hydraulic system.

5-... according to its kinematics (stationary, moving (rotating/reciprocating)).

- During an engine design, a piston, a crankshaft and a cylinder should be handled differently. That is because the kinematics of each element has major effects on its design process, for example, dynamic balancing due to a crankshaft's rotation and wearing of the inside surface of the cylinder due to the friction of the piston rings.

For simple mechanical design applications, one of the suggestions according to the first classification can be shown in Figure 6.3.

At this point, a hierarchy of the tangible objects (elements) which are useful in the design domain has been built up. Now, it is required to join these elements to create a desired design. In mechanical design, connection means exchanging force and torque between elements. One way of implementing such a connection in object-oriented programming is to define a connection as a class. This will be an abstract class and the best definition for its subclasses would be to classify them by non-zero components of force and torque applied between the two elements.

As mentioned, the connection of two elements means exchanging force and torque between them along the three force components (Rx,Ry,Rz) and three torque components (Mx,My,Mz) in three dimensions design. Then by assuming these components as zero or non-zero, different subclasses of the connection can be defined. For example two independent elements have all their components as zero (no connection at all), or in Fixed connection all components are non-zero. For other suggested connection's subclasses the reader is referred to Figure 6.4.

There are some problems in the above classification. Friction forces in the connections should be carefully considered. Although friction produces force, this force is somehow different from other forces. Different kinds of friction should be considered in different situations. In this

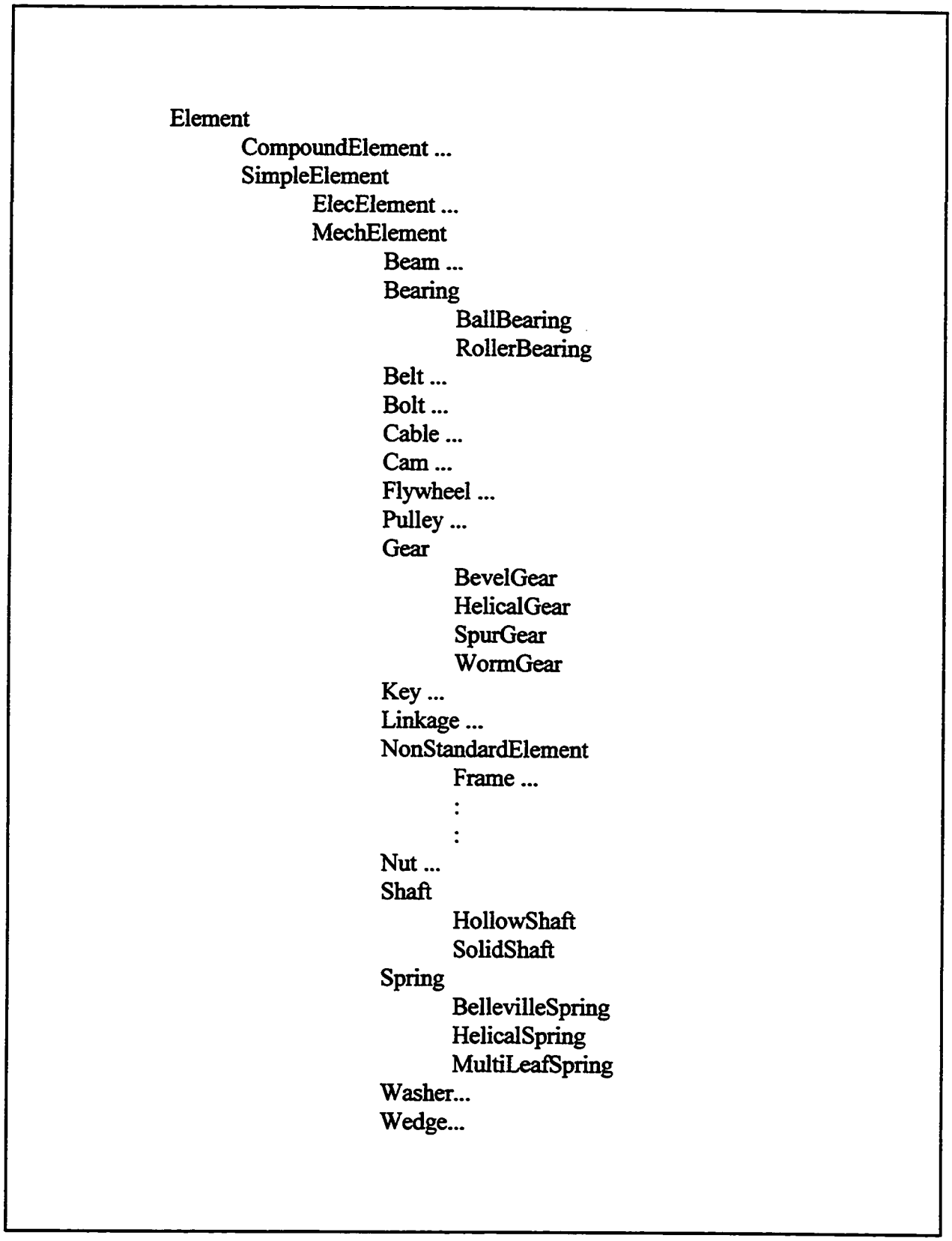


Figure 6.3 : Mechanical Elements Hierarchy

Connection		
BallJoint	(BJ)	$M = 0, R \neq 0$
Cable	(CB)	$M = 0, R \neq 0$
Fixed	(FX)	$M \neq 0, R \neq 0$
GearToGear	(GG)	$M = 0, R \neq 0$
JournalBearing	(JB)	$M_y = 0, R_z \neq 0$
RoughSurface	(RS)	$M = 0, R \neq 0$
SmoothSurface	(SS)	only $R_z \neq 0$
ThrustBearing	(TB)	only $M_y = 0$
Wheel	(WH)	$M = 0, R_x = 0$

Figure 6.4 : Connection Class and its Subclasses with Zero and Non-zero, Force and Moment, Components

work the friction forces are not considered.

The second problem in the above classification is the elements which are used only as connection elements (e.g., Key for connecting Gear to Shaft in Figure 6.5 a). If these elements were not to be assumed as Mechanical Elements, the objects relations diagram will be less complicated because there are only two elements (e.g. Gear and Shaft) with one connection (Figure 6.5 b). Changing the connection would not effect the rest of the

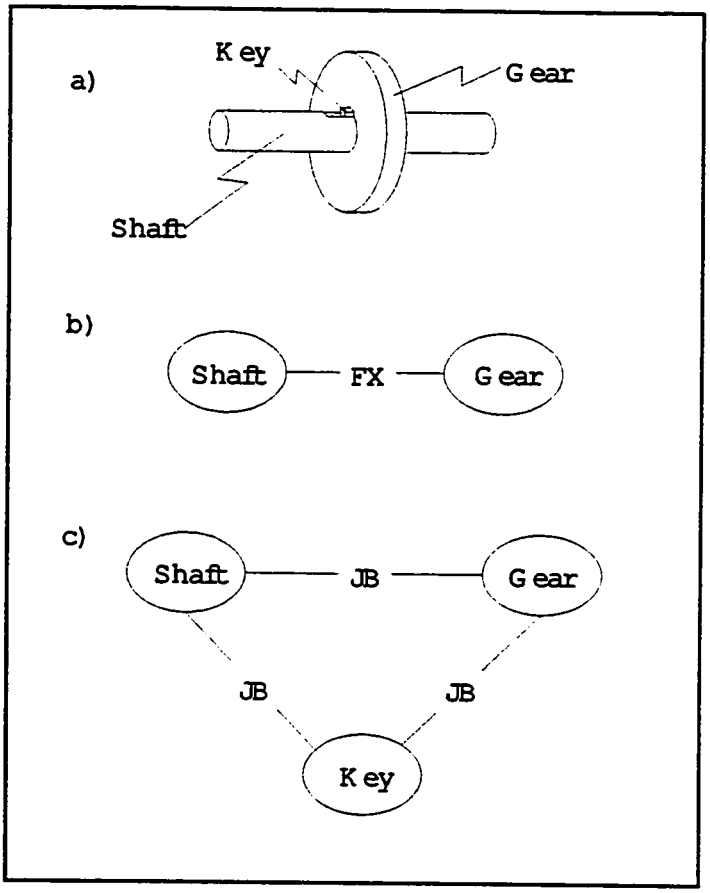


Figure 6.5 : Connecting Element a) a shaft and a gear connected by a key b) the key represented by a fixed connection object. c) all three components considered as objects with their connections.

system. However when assuming the connection as a new mechanical element, there will be three elements with three connections (Figure 6.5 c). Hence, changing the connection now, the connection element and at least two connections will be affected and have to be changed. Having all these problems in this assumption, due to the fact that these simple elements take individual designing processes, connection elements will be assumed to be mechanical elements.

A further complication arises due to magnetic or gravity connections which are not actually physical connections, however there is a force exchange between the two elements (e.g. in the electrical motors).

6.4.1 Ownership vs. Inheritance

All the knowledge of mankind is grouped into classes or sets and the same applies to engineering design. To identify classes, names are associated to them. These classes are grouped into superclasses and the superclasses into even bigger ones. This knowledge is organized hierarchically, and it involves an understanding of the inclusion relationship of all these classes and an awareness of various properties shared by all members of particular classes.

The inclusion relation on a set of classes is very important, and some interesting questions arise when incorporating it into an object-oriented system. The fact that "A spurgear is a gear." expresses that the class of spurgears is a subclass of the gear class. Therefore, the data structures used to represent inclusion relations are often called "IS-A" hierarchies in some AI languages like Common Lisp. However, in Smalltalk, these kinds of data structure are called "inheritance" hierarchy.

Although it is the most important relationship in object-oriented languages, the inheritance relationship ("inheritance" in Smalltalk and "IS-A" in some others) is not the only relationship to create connection among classes and objects in these languages. There are other relations such as "has," "ownership" and "pairs" and so on. Some of these relations can make a hierarchy of classes or objects, but some others create some kind of network connection among classes or objects, while others just link some of these class or objects.

One must beware of some relationships that are similar to inclusion but are really quite different. For example, "spurgearA is a spurgear" does not really mean that the spurgearA class is a subclass of the spurgear class. Rather, it states that the particular object, spurgearA, is a member of the class of spurgears. This relationship is called "membership" relation in some AI languages, while it is called "instance" of a class in Smalltalk. One would represent these relations in set notation as follows:

$$\text{SpurGear} \subseteq \text{Gear}$$
$$\text{spurGearA} \in \text{SpurGear}$$

In this research, not all of the relations used in an AI language like Common Lisp represented as links will be discussed. In a machine design, which deals with systems like machines containing subsystems in several levels, (e.g., an automobile in Figure 6.6) the ownership relation is important. In Smalltalk, inheritance and instance relationships are well defined among classes and objects, while ownership relation is not considered as a relationship in Smalltalk. The complicated case of ownership like in Common Lisp, where dynamic ownership was assumed in some cases like buying and selling, is not considered in this work.

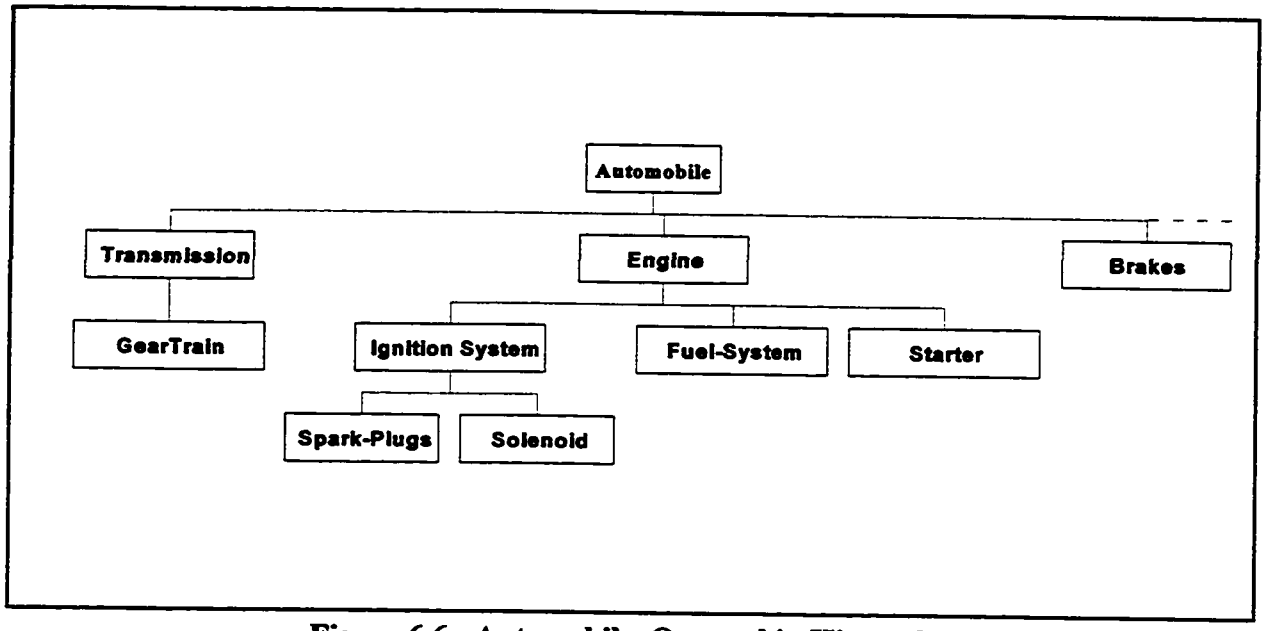


Figure 6.6 : Automobile Ownership Hierarchy

The goal here is to make a hierarchy of the machine components by using the ownership relation. The simplest approach to build the ownership structure is to define an instance variable called "owner" for all classes that will be used in the CAID system. With this variable, each object in CAID will know its owner, but not the objects which belongs to it. Figure 6.7 shows this ownership relation in the CAID system The Object Modeling Technique (OMT) notations [30] is used in this figure which is explained briefly in Appendix A. Although the 'owner' instance variable is enough to make the ownership hierarchy and search in it from bottom to top, it is difficult to deal with the cases that need data from different objects in different levels of the hierarchy. To overcome this difficulty, some sets as instance variables are defined in classes where applicable. In one perspective, the "mountedComponents" is an instance variable of the Shaft class and contains the names of the components mounted on the shaft.

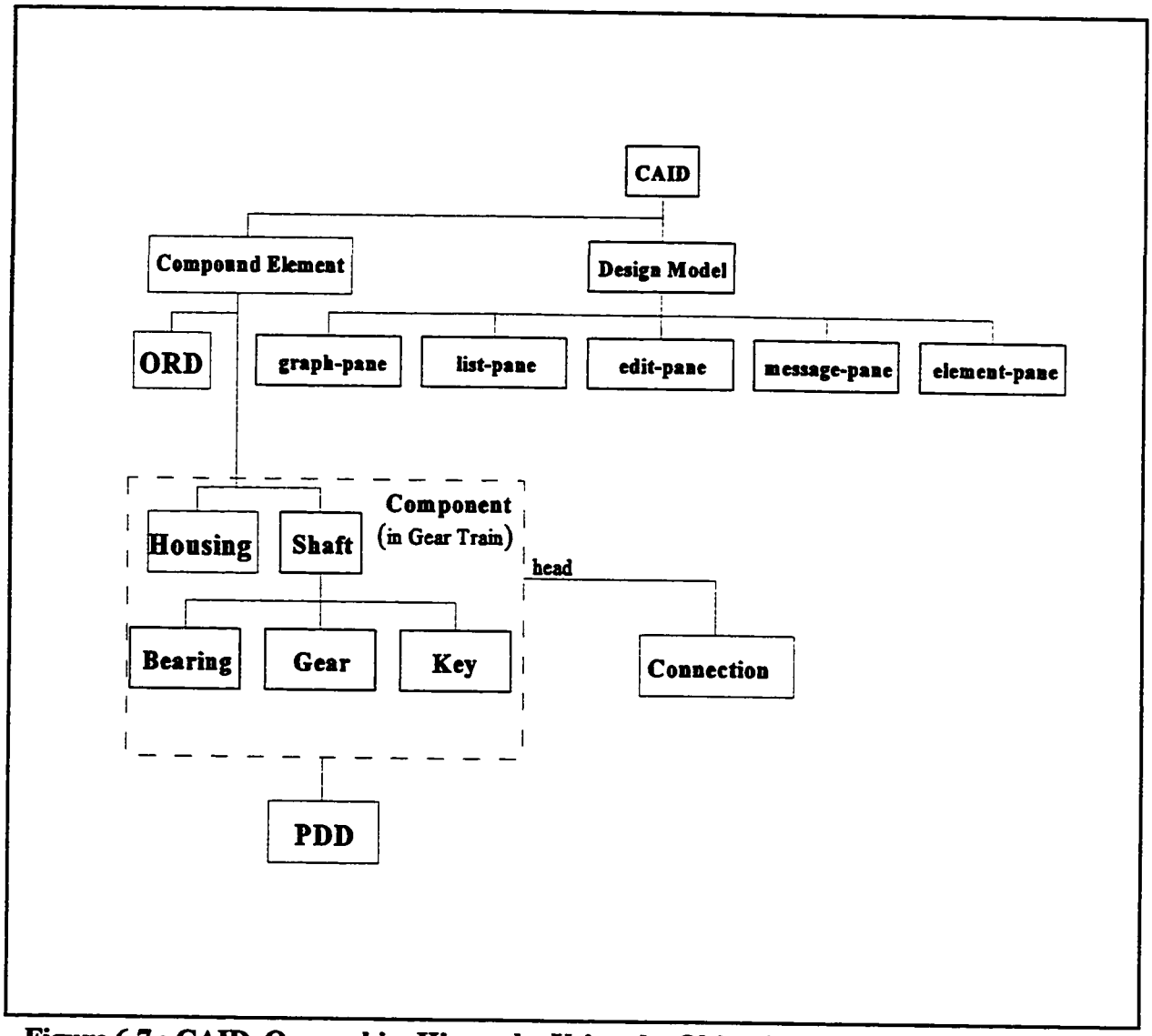


Figure 6.7 : CAID Ownership Hierarchy Using the Object Modeling Technique (OMT)

There are several advantages for having the ownership relations among the objects of a design model. For example, with the ownership relation, a design case is as a integrated entity, which makes it easier to deal with several design cases at a time and process them in parallel, and store these design cases in any stages of the design process. Another advantage in the ownership relation is that unused components can be disposed of and free the memories occupied. Although that is done automatically by "garbage collection" in Smalltalk, when the CAID system gets very complicated,

it will be better to give the control of freeing the memory to the design engine rather than to Smalltalk. This is not only efficient but also it makes it possible to implement the CAID system on other object-oriented languages which does not have this feature.

6.5 CAID abstract class and its subclasses

As was mentioned, inheritance hierarchy can not contain all aspects of the knowledge related to classes and objects relations. Considering only the inheritance relation when creating the classes for CAID, these classes will scatter among hundreds of the classes of Smalltalk. To make it more organized, all the classes needed in CAID are created as subclasses of one abstract class called the CAID class (Figure 6.8). It does not matter if they inherit anything from the CAID class. The only thing that is important is that they should belong to the CAID system. The same rule can be applied on subclasses of these subclasses. For example Diagram and Component are two elements of Objects Relations Diagram, so they will be subclasses of ORD class that represents Objects Relations Diagram. These two subclasses do not have any common protocol to be implemented in ORD class.

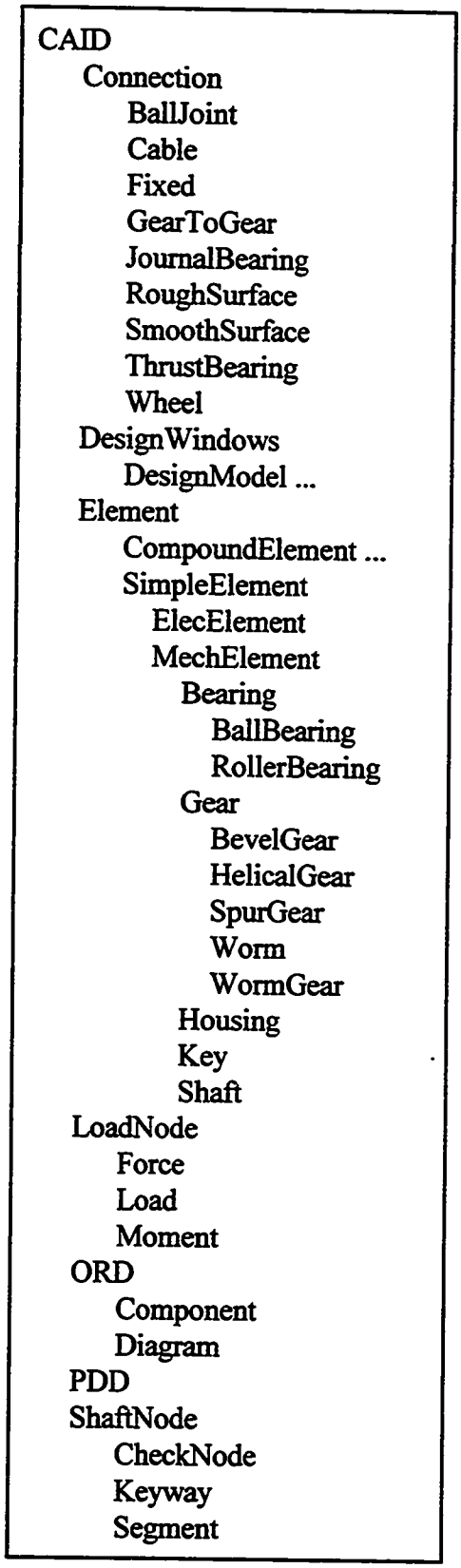


Figure 6.8 : CAID Subclasses

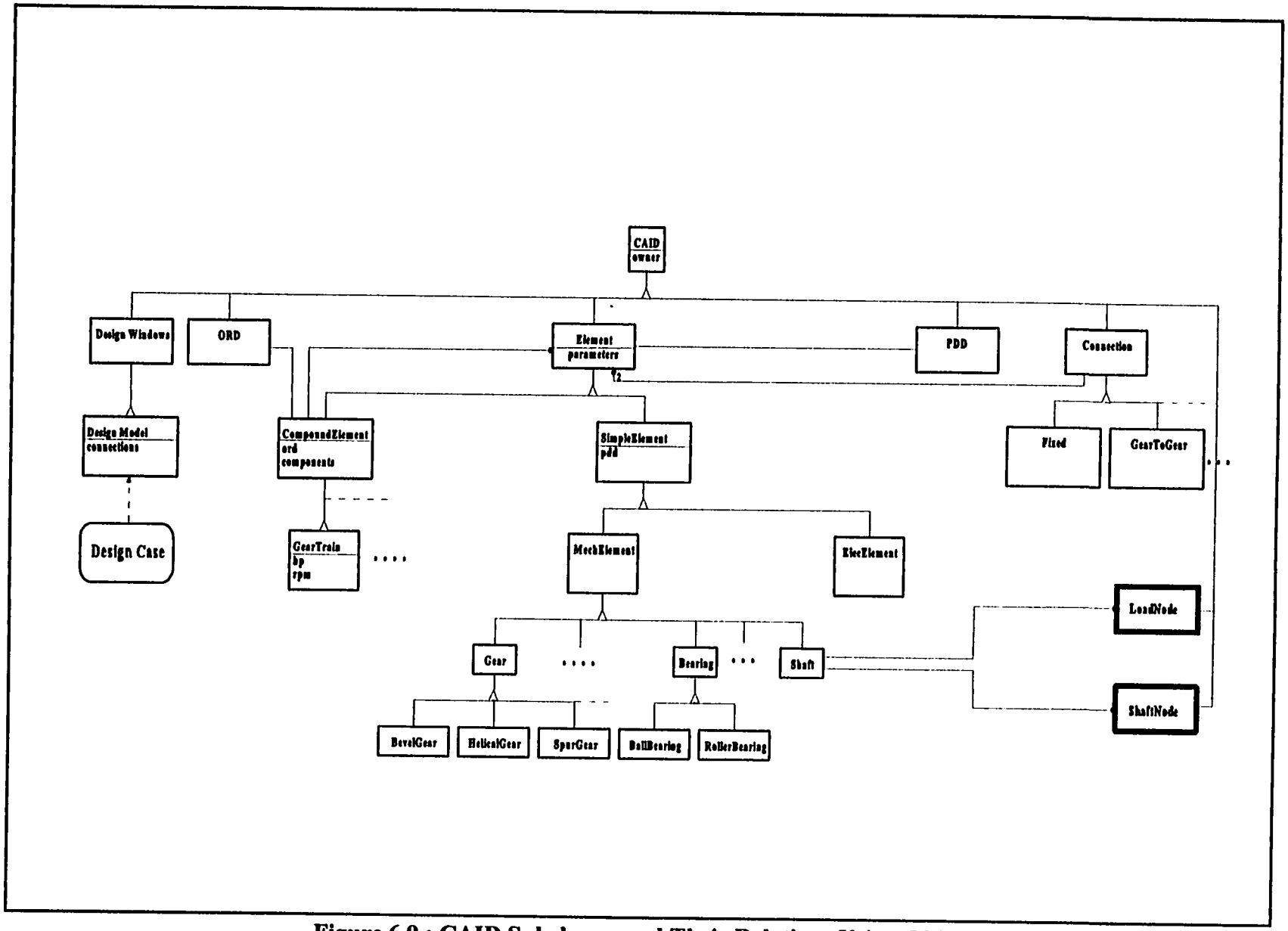


Figure 6.9 : CAID Subclasses and Their Relations Using OMT

Hence, the subclasses of the CAID class are a set of classes that work together to make the CAID system. An instance variable named "owner" in the CAID class is defined and all its subclasses will inherit this instance variable from this class.

In order to show the relation between these classes, the object modeling technique (OMT) is used to create the object diagram in Figure 6.9. As can be seen from the figure, each Design Model has a compound element with an ORD and several Elements which are called components. Each Element has a PDD, and each Connection relates two elements. Each Shaft can have several LoadNodes and several ShaftNodes. The Design Model will be elaborated on in another section.

ORD and PDD classes are used as models for simulating the object relation's diagram and the part design diagram of the element, respectively. These classes are discussed in chapter four and will be elaborated on in the following sections. LoadNode and ShaftNode are superclasses for two sets of subclasses created to ease the shafts' design. LoadNode's subclasses simulate loads like concentrated force, moment or any other distributed load on the shaft length, while ShaftNode's subclasses simulate important points like steps, keyway or any other point on the shaft length which changes the shaft section area or shape. Segment is another subclass of the ShaftNodes which simulates moment segments of a moment diagram (Figure 6.10). Instances of this class can be used

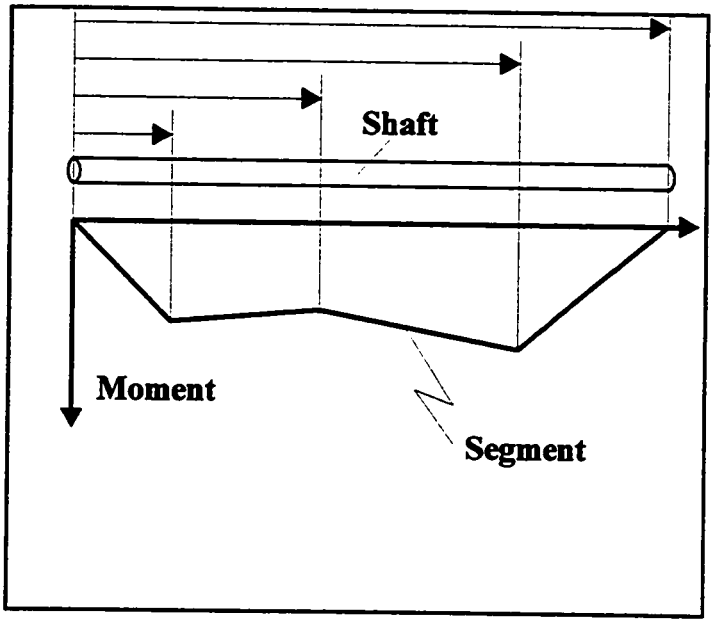


Figure 6.10 : Moment Diagram Segments

for simulation of the distributed loads which consist of several segments.

The connection object is the communication vehicle between two connected components. Each connection has two pointers (called the head and the tail) which each of them refers to one of the connected components (Figure 6.11). There are other instance variables in each connection object such as headForces and tailForces which refers to head and tail forces. These forces are calculated according to the types of connected components. The owner of the connection object is its head component. This eases the process of the calculation of the transmitted rpm and horsepower from input component to the output component.

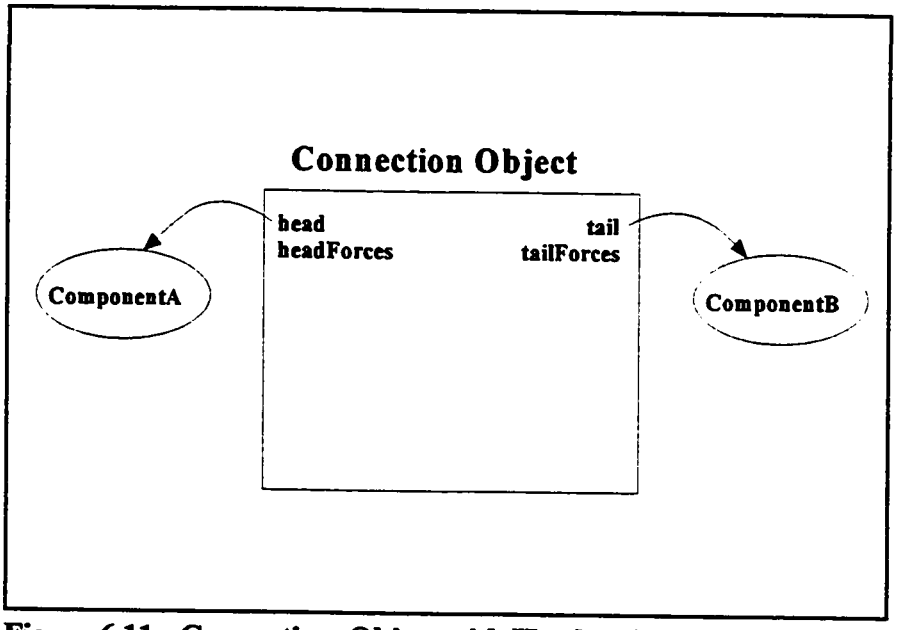


Figure 6.11 : Connection Object with Head and Tail Components

Distinguishing between the outside view and the inside view of a class is important in its construction. Emphasizing the abstraction, and hiding the structure and the secrets of the class behavior is provided by its interface which represents the outside view of the class. The main aim of the interface is the declarations of all the operations applicable to instances of that class. Unlike the interface, the implementation of a class is its inside view. It includes the secrets of the class behavior and the implementation of all the operations (methods) defined in the interface of the class.

Having established the concrete interface of each class and object important to the design at

a given level of abstraction, the implementation of these classes will have been completed by the end of the implementation stage of the design process. Therefore, the products of this stage include the refinement of the class structure of the system, and in particular the completion of each important class.

6.6 Objects Relations Diagram (ORD)

It is assumed that the designer knows how to create the design layout. The previous design layouts and the design engine of the CAID system will help guide a user to create the object relation diagram (ORD). Figure 6.12 represents the state diagram of the object relation diagram.

Starting a new diagram, the design engine will provide the user with a list of design elements which has been identified in the previous step. The user can create as many instances of each element as required in the design by pointing to the name of that element in the list. Each time that the designer creates an element, the system will suggest some connecting elements which that type of created element requires and which have not already been created. For example, in a gear train design, if the designer creates a shaft, the program will suggest two support bearings and one mounted gear. In some cases, it can even suggest the type of the bearing or the gear, such as spur mate gear, for a spur gear. If the designer has a special configuration which does not match the design engine suggestion, he can create a new element and connect it to the related element.

Having created all the elements for the system, the designer has to choose the types of connections from a list which depends on connecting elements type. By completing the objects relations diagram, the designer can move forward to the parametric design stage by identifying the critical element (e.g. input shaft in gear train design). This diagram will be frequently used by the

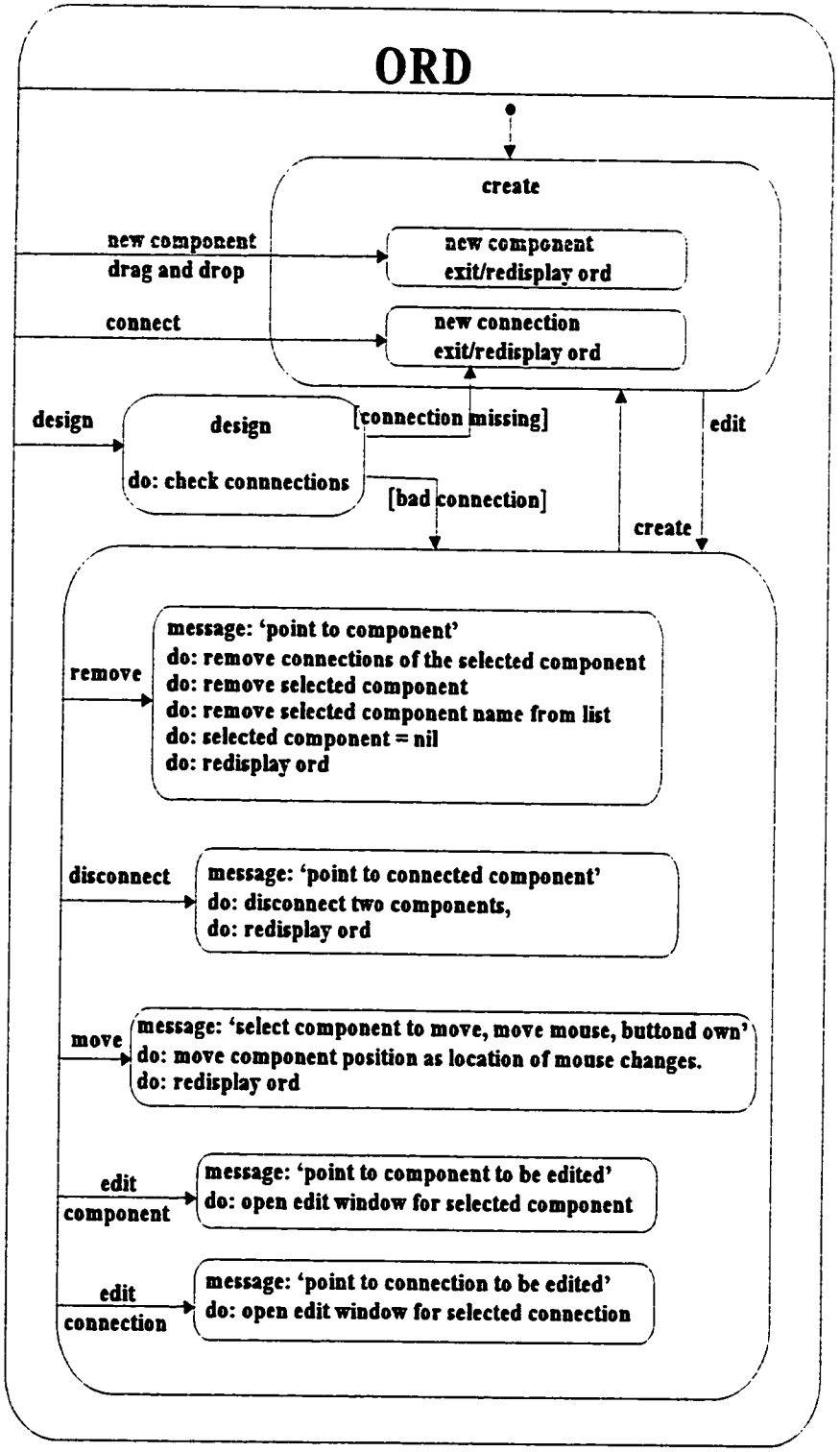


Figure 6.12 : State Diagram of ORD

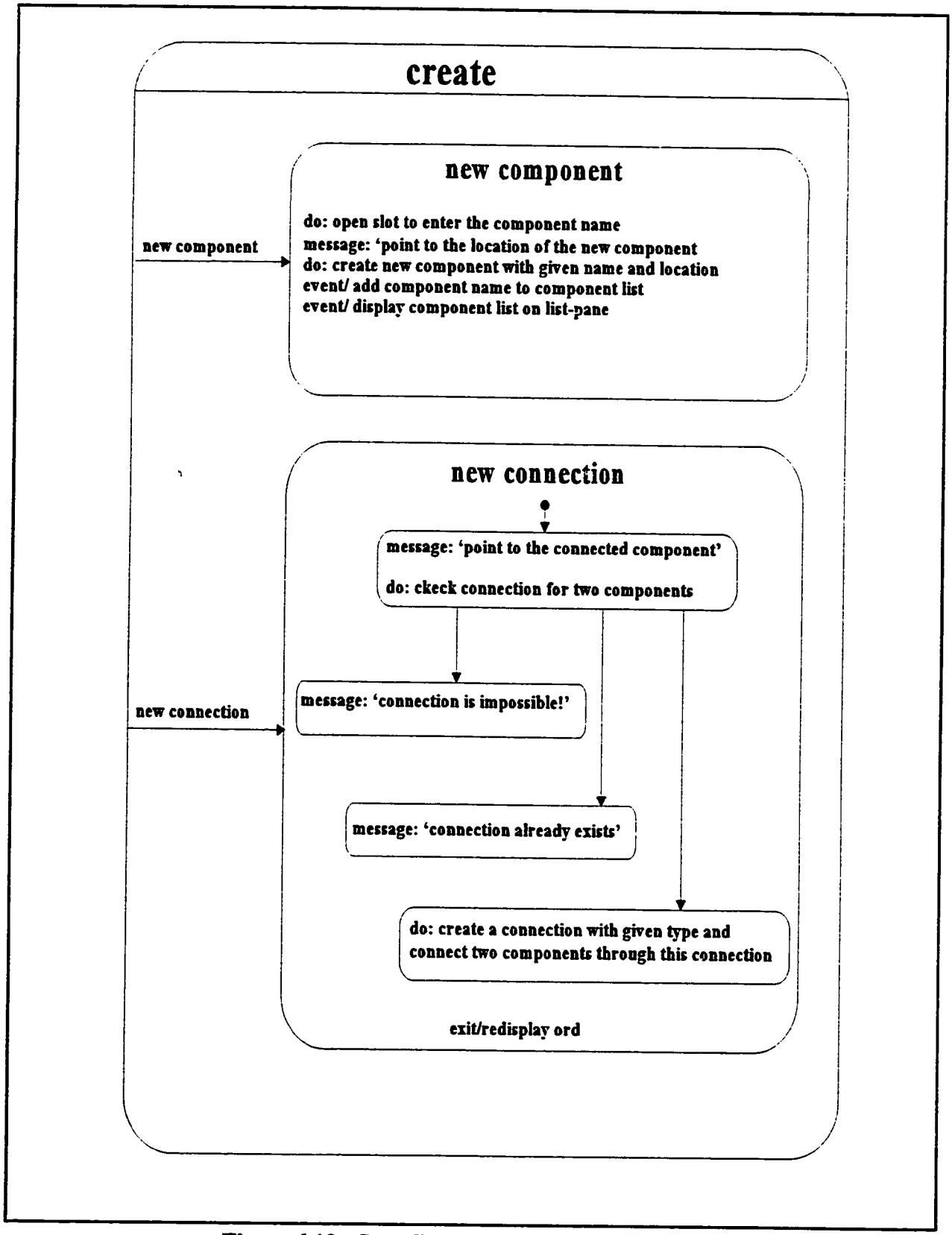


Figure 6.12 : State Diagram of ORD (continued)

design engine during the design process, and it is a part of the design model which will be described in another section.

6.7 Design Model

The design model consists of a complete list of components, their relations, and input/output parameters. Located within the design model are design procedures or methods. These methods do not appear on the model window. Figure 6.13 represents a gear train design model.

DesignModel has been defined as an abstract subclass of DesignWindows. By sending the "openOn:" message to any design model an application window will be created with a topPane and several subpanes as its children. These subpanes can communicate to each other and synchronize their actions.

The design model acts as a template for the design case. In the design model the design procedures are defined precisely, but the input parameters are unknown and the components list is empty. In the design case, which is a copy of the design model, the user defines as many input parameters as possible. Assuming that the layout of the design is known, by continuing the design process interactively the user creates the components which the design needs. These components are the instances of the Element subclasses (i.e. Shaft, Gear . . .). The user might create several design cases and run them in parallel for comparison purposes. These design cases can also be compared with the previous design solutions which have been stored.

Figure 6.14 represents the Object Model of the Design Model using OMT. The graph pane can either represent objects relations diagram (ORD) of the model or the part design diagram of a component. A list of the element names represent the classes of the elements which are used in this

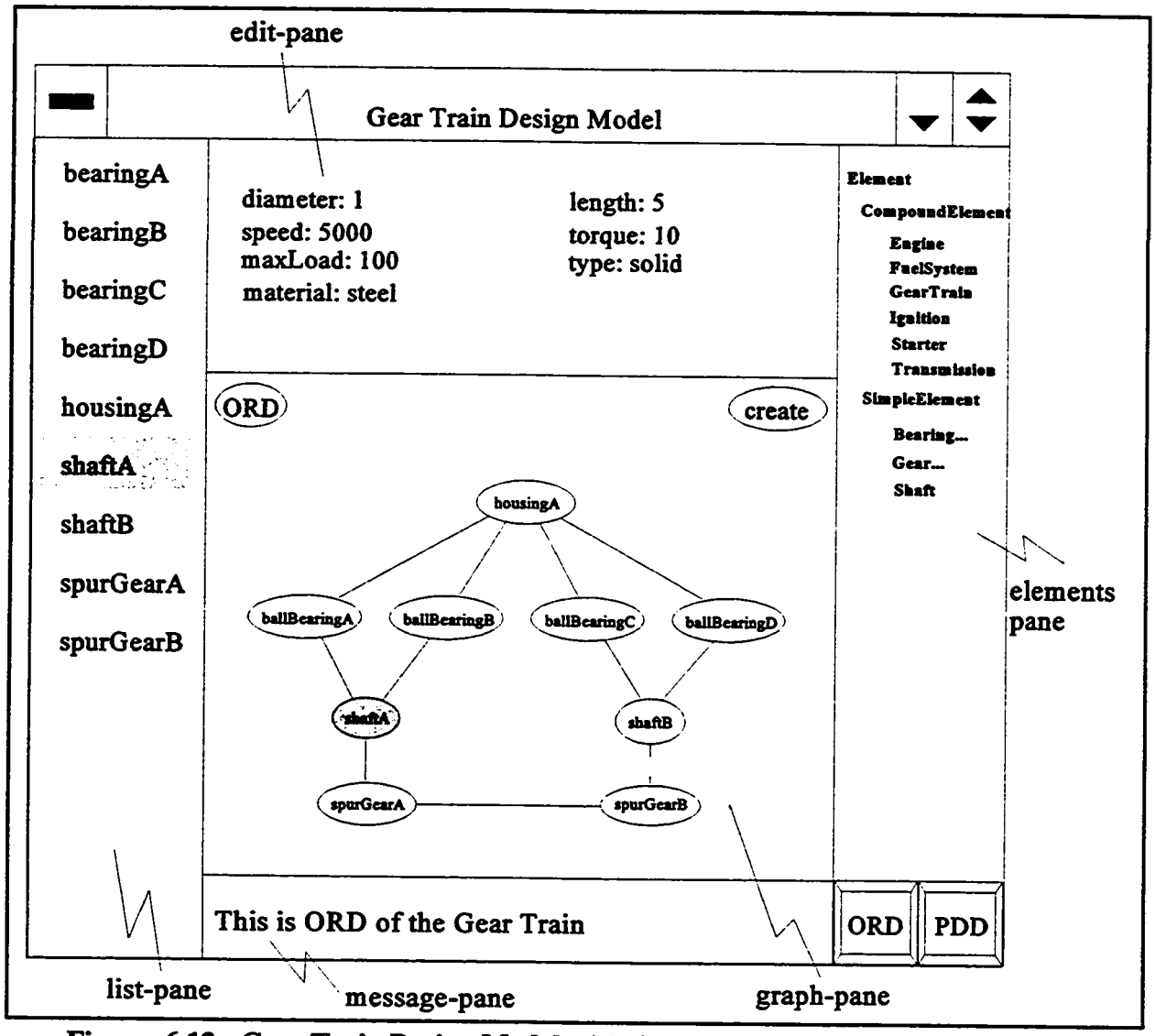


Figure 6.13 : Gear Train Design Model with Component List and Their Relations

kind of models. By pointing to a name in the list the user can create an object which is an instance of that class. The graphical representation of this recently created object will appear on the objects relations diagram, and its name will be added to the list of components in another pane.

Having created the component, the user continues to define the relations among the components by creating the instances of related Connection subclasses accordingly. The design

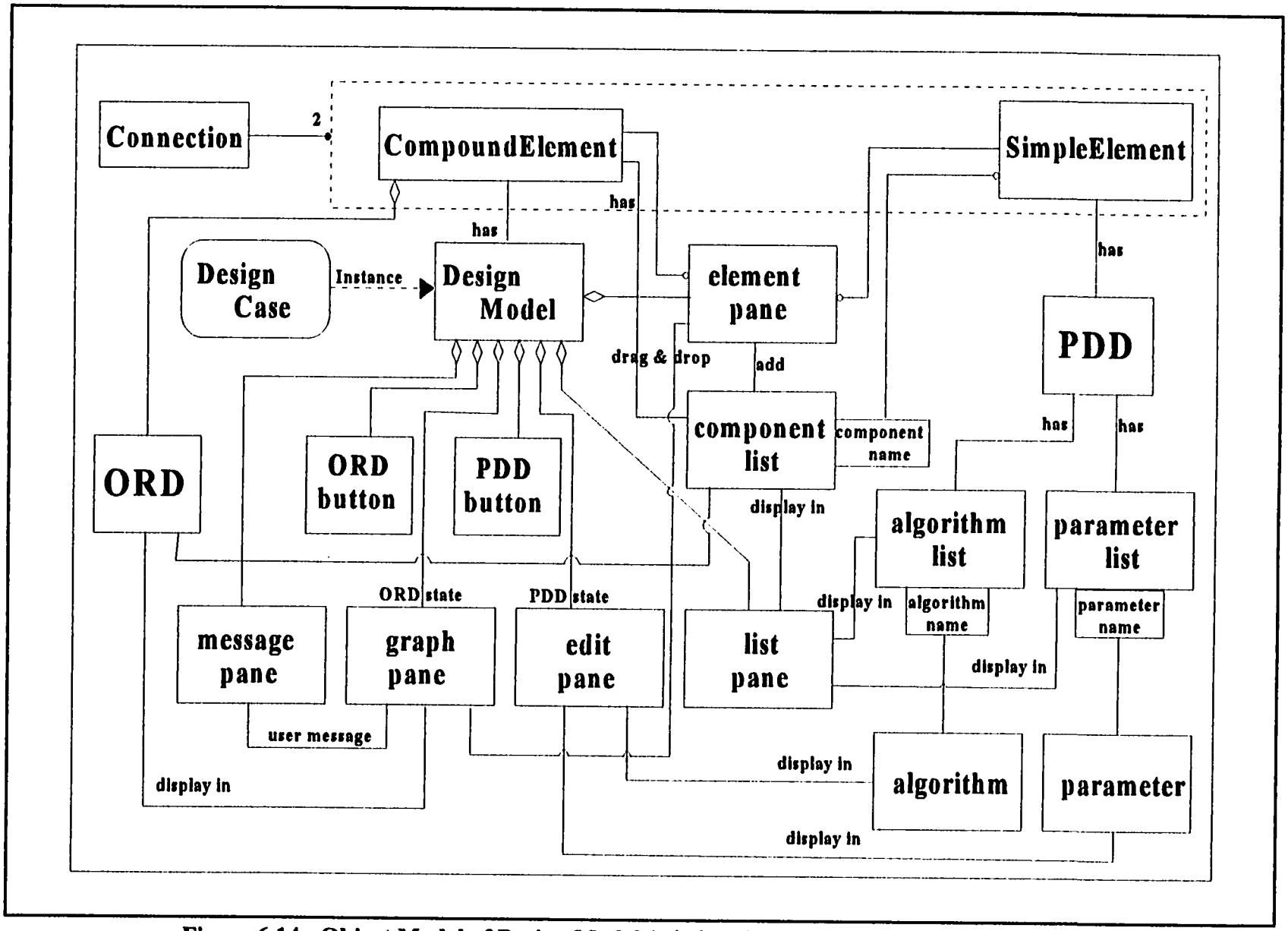


Figure 6.14 : Object Model of Design Model (relations between objects according to OMT)

process will be continued by calling appropriate design procedures (i.e. methods in Smalltalk) defined in the design model. Other procedures in each component will determine the values of their variables which were unknown during the time of component creation. The process of determining the variables values for each element is illustrated by its part design diagram (PDD).

PDD could be an independent class or integrated methods in each component. If it is an independent class, then there will be an instance variable namely "pdd", for each element, which points to a real PDD. In the previous sections, the application of each element was assumed as a criterion for classifying the element hierarchy. Because the design process of an element directly depends on its application, the PDD of each element will be encapsulated in its own structure.

The dynamic relations between different parts of the design model have been shown in Figure 6.15. If the user wants to know some of the parameters of the object, it is easily done just by pointing at the object's name in the component list, and change the state of the list-pane to the parameter state. Then, the parameters of that object will appear on the Edit pane and the selected parameter value on the edit-pane. The user can provide the initial values to these parameters or make any change in their previous values. The design engine starts the design process by knowing the input parameters, and as was mentioned in the design engine section, it continues to design an object, one step at a time, by revealing its part design diagram, which is hidden in this window.

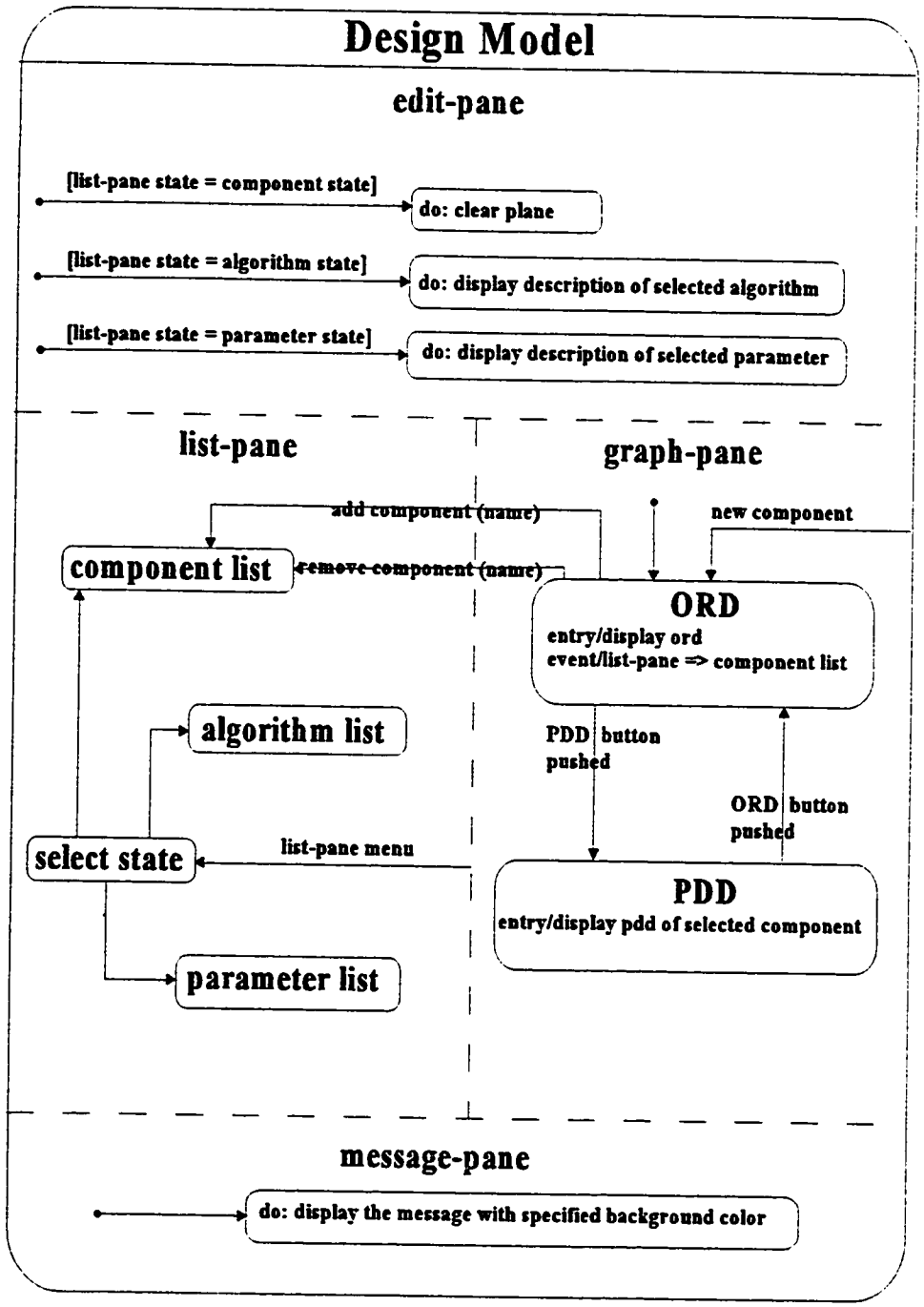


Figure 6.15 : State Diagram for Design Model

6.8 Part Design

Parameters can take specific values to satisfy the requirements of a specific design. Conversely, parameters can be varied over a range of values to satisfy the individual requirements of a range of designs. If the parameters were completely independent of each other, their values could be chosen at will. However, parameters characterizing machine elements are usually interrelated by equations derived from physical principles of geometry, mechanics and strength of materials. Parameters may also be related or constrained for purposes of standardization, or to take advantage of knowledge gained from previous experience with such machine elements. In the example of the gear train design, as one might expect, the load and deflection parameters are related to the geometric and material property parameters by equations derived from principles of strength of materials. In the presence of parameter relationships, values for parameters are no longer independent. In solving a specific problem, parameter values must either be known in advance or must be determined so as to satisfy the parameter relationships.

Typically, in a design situation, parameter values of geometry and material are to be determined from given values of load, deflection and stress parameters. However, a complementary situation also arises in which machine elements or a machine element design already exists. Here the geometry and material parameters already have fixed values and one might be interested in determining limiting values for load, deflection and stress parameters. Further, it is also possible that given parameter values of a specific design do not satisfy the relationships for any values of the other parameters. In any of these cases, however, once parameters are identified and parameter relationships are derived, machine element design becomes a problem of finding appropriate values for parameters while satisfying parameter relationships.

Machine element design, regarded as parametric design, can thus be expressed as a two step process:

1) Determine the parameters of the structure and the behavior that can be varied for the machine elements, and derive relationships between the parameters from physical principles, standards and previous experiences.

2) For a specific design, determine appropriate values for unknown parameters from known parameter values using the derived relationships between the parameters. In some cases, this step may require iteration with trial and error to determine parameter values. In some other cases, no solution may exist, or given parameter values may themselves be inconsistent with the derived relationships.

In order to deal with the process of parametric design of machine elements in a systematic and efficient manner, there should be a description of the parameters, the relationships and the conditions under which they apply. Traditionally, these descriptions have been carried out through algebraic equations and informal language (English) descriptions. Graph representations, used by some researchers, are more effective for understanding the parameters and parameter relationships involved in the design of a machine element. Further, design diagrams bridge the representational gap that usually exists between Step 1 and Step 2 of the above two step process and provides a standard procedure for executing Step 2.

6.8.1 Part Design Diagram

Part Design Diagram (PDD) which is explained in Section 4.2.1, will be used to design machine elements. As mentioned, the diagram contains variables which may be known or unknown, and algorithms which are procedures. It is not necessary that all the algorithms be explicit. In order to solve some part of an algorithm the design engine can even ask for the user's help. Although the exact form of the algorithm relating the variables may not be known at the beginning, the knowledge engineer may believe, by any logical reasoning, that there is a relation among those variables. The only prime requirement is that the unknown variable can be determined, by some operation, using known variables.

This diagram is capable of showing whether connections from the known to unknown variables are sufficient, and of indicating those that would be missing. Figure 6.16 represents the state diagram of the PDD. The supposed sequence of the operations can be derived easily by identifying the knowns and the unknowns on the diagram. Two different solution search methods, namely backward and foreward chaining, could be used to order these operations. The brief descriptions of these two methods are as follows:

Backward: Starting from an unknown parameter, there will be different algorithms related to this variable (unknown parameter). There are two criteria for choosing the best algorithm to determine unknown parameters.

1-Simplest algorithm (which is difficult to define and locate).

2-Algorithm with least unknown parameters.

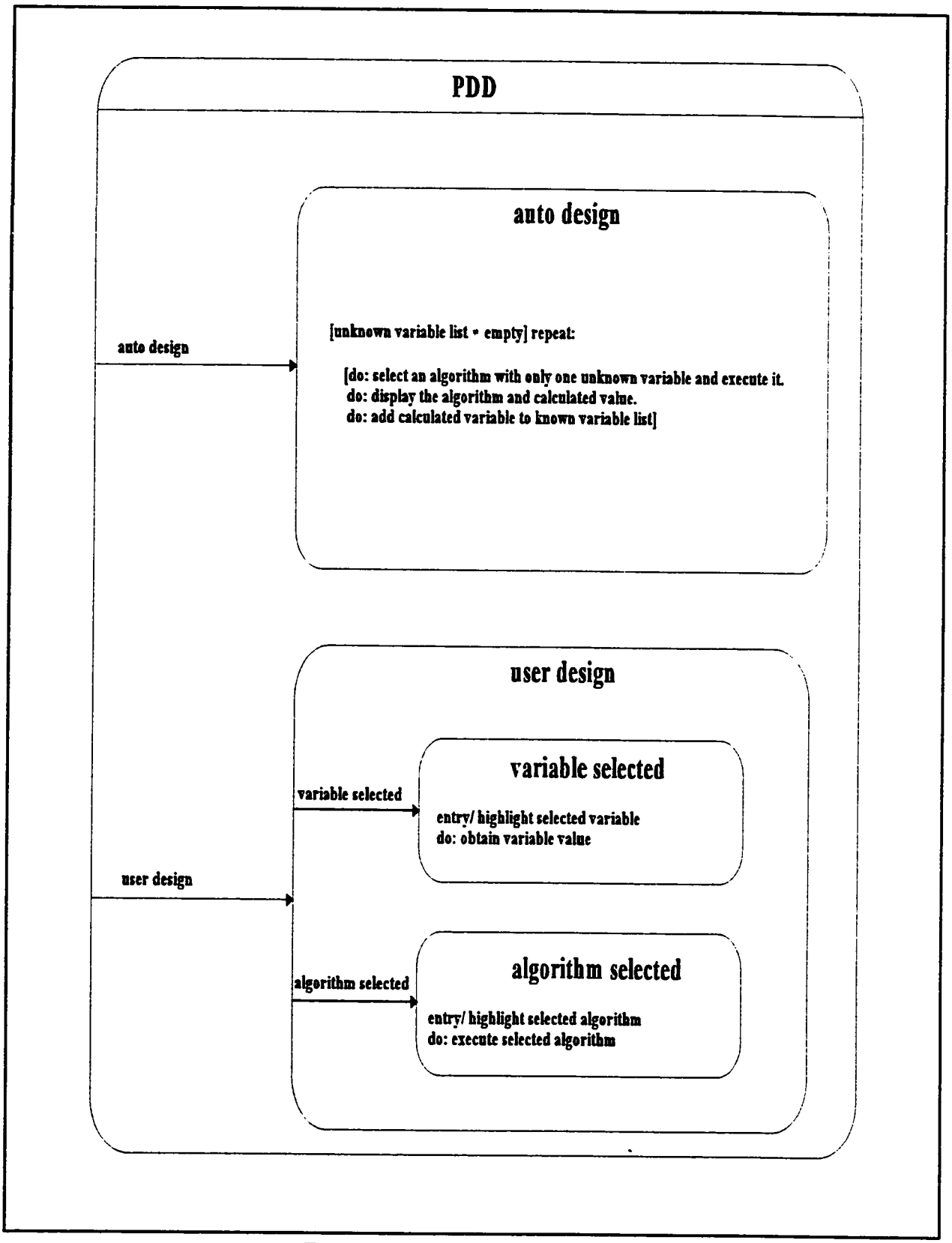


Figure 6.16 : PDD State Diagram

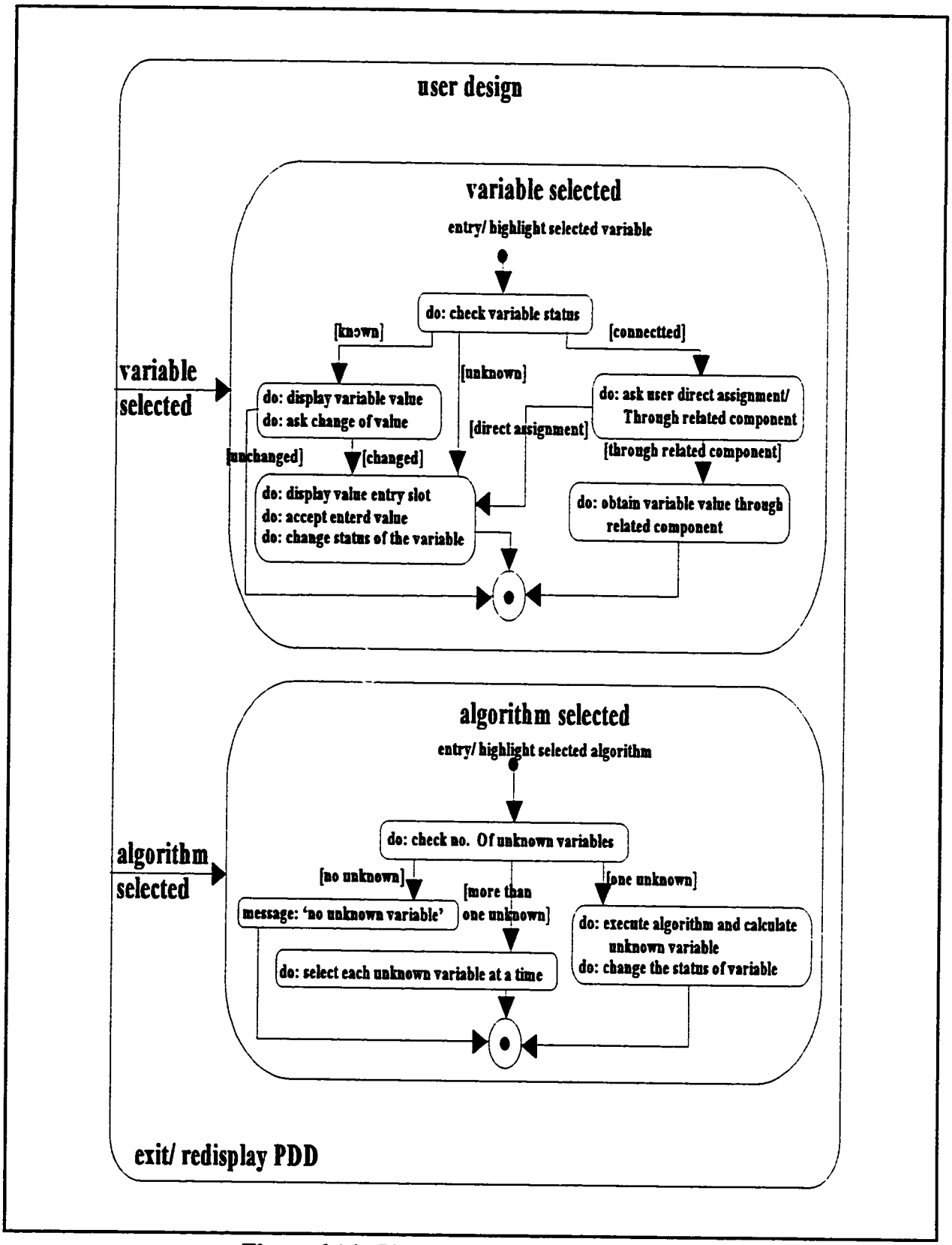


Figure 6.16 : PDD State Diagram (continued)

Forward: Starting with an algorithm with only one unknown parameter, and determining that parameter. Continuing this process until all parameters (variables) are known. Although the values of all parameters might not be of interest, in this method there is a high possibility that all parameters will end up being calculated. This method seems good enough for parts with less than 20 parameters. In this work only the second method will be assumed, because parts are simple. For complicated parts it is better to use a combined approach (Forward and Backward). It is possible that there will be no algorithm with one unknown. In this case the design engine has to query the user for suggested values for some parameters or obtain those from previous designs.

Using the forward method, and assuming a set of known variables, the algorithms with one unknown variable (their other variables are known) are determined and executed individually. Following the individual calculations, each of these calculated variables will be added to the known-variable set. After a cycle of all related algorithms executions, the set of known variables is updated. At this point, the design engine will check all algorithms again, continuing to trace the cycle till the completion of the requested unknown variables.

6.9 Design Engine

As mentioned, the design engine is the driving mechanism for completing a design. Although the design engine can advance the design cycle through all the stages implemented in the design model and other modules of the CAID system, the user has control and is able to direct the design engine in its decision making process. The design engine is a communication mechanism among all the modules of the system.

At the first step, the design engine obtains the layout of the design from the user of the CAID system. It gives schematic views of the previous designs which are their objects relations diagrams.

The user modifies and/or combines them and gets a layout for the design. If there is no previous design, the design engine starts by asking the designer to create an element and after creating an element it will suggest some connecting elements which that type of created element requires but was not already created. Having the layout of the design, the design engine creates instances of its elements and connections.

There are three independent design cycles, namely a preliminary design cycle which uses rule of thumb to suggest values for variables, the second cycle, "complementary cycle" uses formal equations to calculate values of variables. The final cycle uses advanced analytical methods and programs to carry out the design analysis. This latter cycle is out of the scope of the current work. Only the two first design cycles will be considered and included in this work .

Input data that define the nature of the design problem at hand will be passed on by most of the elements in the design, so the design engine will consider this process before any other possible action which the design engine can take. This process usually proceeds by introducing one of the critical elements (as input element) either by the user or by the system itself. During part design, it might be necessary to jump from one node (PDD) to another. If in these jumps the design engine reaches a node which it had already passed and it was looking for the same uncalculated parameter, it is in a loop. The design engine will in this case consult the designer to find a way out of this loop.

The design cycle continues by completing one element (starting with the input element) and proceeding to other related elements using the interface variables. If in some cases there is no rule to assign a value to a variable and the system can't proceed without assigning any value to that variable, the design engine will ask the user to suggest any value or choose a value from a suggested list.

After the first iteration (preliminary design cycle) using the rule of thumb for assigning value to variables, the second iteration (complementary cycle) will proceed in the same manner except that formal equations rather than rules of thumb would be used.

Chapter 7

How CAID Works

7.1 Introduction

In this chapter, the detail of how the CAID system works will be discussed. In Section 7.2, it will be shown how a new system can be created from predefined subsystems. In Section 7.3, the method of storing the completed design cases will be considered. In Section 7.4, the measures which are taken to protect the CAID system will be mentioned. Finally in Section 7.5, the design process of a gear train components will be analyzed.

7.2 Creating New Systems

At the beginning, an icon in the center of the screen will shows the readiness of the CAID system. A small example of this icon is shown in Figure 7.1. The designer can point to this icon and a questionnaire window with an entry slot will be opened, and the designer should enter the name of the new system to be designed. Although this new system will be constructed using predefined CompoundElement subclasses, its name should not be as one of these subclasses. After entering the name of the system, a design model, with a creation mode, will be opened with the system name at its top bar. All panes will be empty except the element-pane which represents predefined elements in hierarchy form. The designer can choose components of this new system by pointing to those predefined compound and simple elements in the element-pane, then the chosen element can be dragged and dropped in any

Figure 7.1 CAID Icon

location in the graph-pane. This is usually referred to as the creation of the components.

There are some abstract classes which act as superclasses of similar elements (e.g. GearTrain, MechElement, Gear, etc.). These abstract classes do not have instances. If the designer tries to create an instance of these elements an error message will appear on the message-pane. These error message backgrounds are in red color to attract the designers attention. It will also blink once to show the most recent message, because the message will stay in the message-pane until the next message appears.

Each component which is an instance of the Element subclasses, will be represented by an ellipse containing the name of the component. The naming of the components will be done at the time of its creation. This name consist of the component's class name starting with a lowercase letter, while it is vice versa in the case of the class name and the letter which will be attached at the end of it. This ending letter is for distinguishing purposes between several instances of the same class. For example, the first shaft will be named as 'shaftA', while the second shaft is referred to as 'shaftB', etc. This naming will be done automatically by the CAID system, but the designer can change it by editing the component in the list-pane. At the end of this mode the designer can enter into the connection mode and connect these chosen components to each other.

Whenever a new component is created, its name will be added automatically to the list of components in the list-pane. Figure 7.2 shows the complete subsystems of an automobile and their connections. When the designer connects the subsystems, the CAID system always checks for improper or already existent connections. In both cases there will be an error message. For example, if the designer tries to connect the starter to the fuel system, since this connection is not already registered by the knowledge engineer, an error message will be given. The registration of the

connection of two subsystems can be done easily by the knowledge engineer.

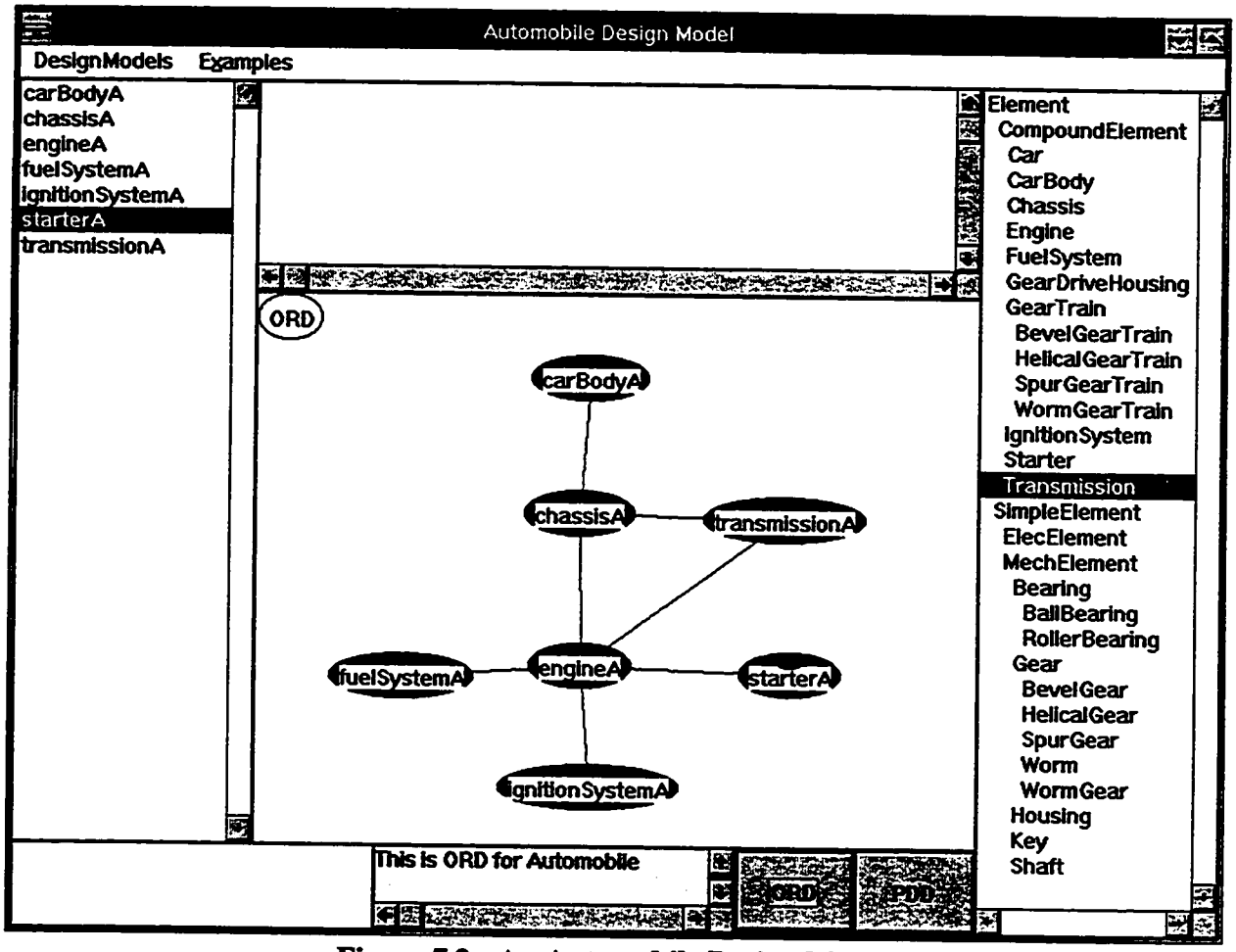


Figure 7.2 : An Automobile Design Model

Because it is not possible for the required space to show all the elements in the element-pane because of the size of that pane, the designer has the option to hide unnecessary elements. This is possible by double-clicking on the superclass of those elements, or selecting a superclass and choosing the hide/show option from the pane menu. For example, when the designer is selecting subsystems of a new system from the CompoundElement subclasses, he can hide the subclasses of the SimpleElement. This makes the selection easier and the designer will not choose a simple element by mistake, because the shown elements are the only selectable ones.

7.3 Storing and Retrieving Design Cases

By connecting the components of the new system, its ord is build up step by step. Completing the ord of the system, the designer can save the system as predefined compound elements in the element-pane. For this process, the designer needs to get authorization from the knowledge engineer. Without the permission password from the knowledge engineer, the designer can not save the new system as a class, but the designer can continue the system design. The process of storing a new system after entering the password is automatic. At first, if there is not a class with the new system name, a new subclass of the CompoundElement class will be created. Then a deep copy of the ord of the new system will be made. In case there exists a class with the same name, the CAID system will ask for permission to overwrite its ord.

The deep copy of an object has special meanings for each type of object in the CAID system. For example, an ord deep copy creates a new ord which has new elements, new components and new connections. Although the names and positions of the components in the graph-pane are new objects but their value will look alike. That means 'shaftA' will be called 'shaftA' in the new ord and will have the same position in the graph-pane. New connections head and tail refer to the components with the same names as the old connections. Hence, if objects of a class require this kind of copying, a method will be implemented in this class to take care of this task. Actually when the CAID system makes a new element, component or connection in the ord deep copying, it creates a deep copy for each of them. Because of the dynamic structure of Smalltalk, the CAID system has to make a deep copy for all of these elements. If the CAID makes a simple copy for some elements of the ord, any change in the copy of the ord will alter the original ord, which makes nonconsistency in the system.

Each component object has three instance variables named: element, name and position. The

name and the position are simple variables and refer to the name and the position of the component in the graph-pane. While the element variable is somewhat complicated. When component refers to a simple element (e.g., a shaft), then the element variable simply points to this simple element. When component refers to a compound element the element variable will be a dictionary referring to several copies of that compound element. The keys of the dictionary are the names of the copies and their values are pointers to these compound elements. In order to simplify the names, the system uses the original name of the component followed by the copy number in sequence. For example, the copies of the transmissionA will be named as transmissionA1, transmissionA2, and etc. There will be the same case for subsystems of the subsystem. That means there will be a hierarchy of subsystem copies, and the designer, after completing the design, can choose one and only one of the completed subsystem designs to complete the design of the system in hand. For example, in an automobile design, the designer may have several copies of transmissions, and each transmission may have several copies of gear trains and housings. After completing the design of all the copies of the subsystems, the designer will choose one gear train and one housing to complete each transmission, and will choose one of the transmissions with other subsystems, like an engine, a starter, and etc., to complete the design of the automobile.

7.4 CAID System Protection

There are some system protection measures which are taken in the CAID system implementation to protect the system from any unexpected changes. CAID accessibility by the knowledge engineer, the senior designer and the junior designer were discussed in Section 4.6. In this section some of these protection measures will be discussed.

The processes which can be done by the knowledge engineer are defined as items of a menu called KE-menu. This menu is hidden from the designer, because the designer does not have permission to make these kind of changes. The knowledge engineer can make this menu to appear in the menu bar by entering the registered password.

Design models are interfaces, and the CAID system is hidden behind these interfaces to be protected from unwanted user changes. When the junior designer uses a design model, only the permitted menus will appear in the menu bar. Other menus like the KE-menu (Knowledge Engineer menu) or Smalltalk menu will appear in the menu bar after entering the correct password by the knowledge engineer. Hence, the junior designer can not alter the CAID system, and the knowledge stored in the system is read-only for the junior designer.

The reason the CAID system checks for existing connections in the connection mode is the repeated connections that cause nonconsistency in the system, and also some elements (for example, two arched members in a framing) may have more than one connection with each other which should be differentiated from a simple connection. In this research, only one connection between each pair of elements is considered.

The knowledge engineer can add any necessary connection or elements to the system. In order to show the possibility of two elements connection, the knowledge engineer can add the type of the connection to the dictionary of ConnectionTypes which is a class instance variable of Diagram class. This process will be done in the addConnection method of the same class.

7.5 Design Process

In order to design a predefined system, the designer will open its design model. Then, selecting one of the subsystems in the list-pane, by double clicking on it, a new design model for this subsystem, will be opened. If the subsystem is a simple element its design continues by using the element's pdd, the process of the simple element design will be discussed later on in this chapter, in the part design section. The design model of the subsystem is the same as the system design model with new components (or subsystems) and a new ord. The predefined elements are presented in the element-pane as before. Opening the design model for this subsystem does not cause the original design model to be closed, or its process to be stopped. The designer can go back to the system design model in the middle of the subsystem design to start another subsystem design process. This can be another copy of the same subsystem and the designer can continue all these design processes in parallel. The designer can terminate any design process by closing its design model if the process is dead-end or its result does not satisfy the design specifications. If there is no ord for the subsystem, the designer can create an ord for it. The designer can even modify the existing ord of the subsystem, but to store the created or modified ord, the knowledge engineer's permission is required.

7.5.1 Part Design

Having broken down the system to subsystems, and subsystems to more subsystems, it will come to a stage that all subsystems will be simple elements which are called parts. There is a pdd in the class of each part which contains the knowledge of the part design. The designer will use this

pdd to process the design of the part. In order to bring up this pdd in the graph-pane, the designer will select the part in the list-pane and by pointing to the PDD button, the pdd of the part will appear in the graph-pane and a new menu named 'PDD' will appear in the menu bar which can be used by the designer to process the part design through the part design diagram (pdd). Figure 7.3 shows the pdd of a spur gear. In this diagram, algorithms are represented by squares, with the algorithm name

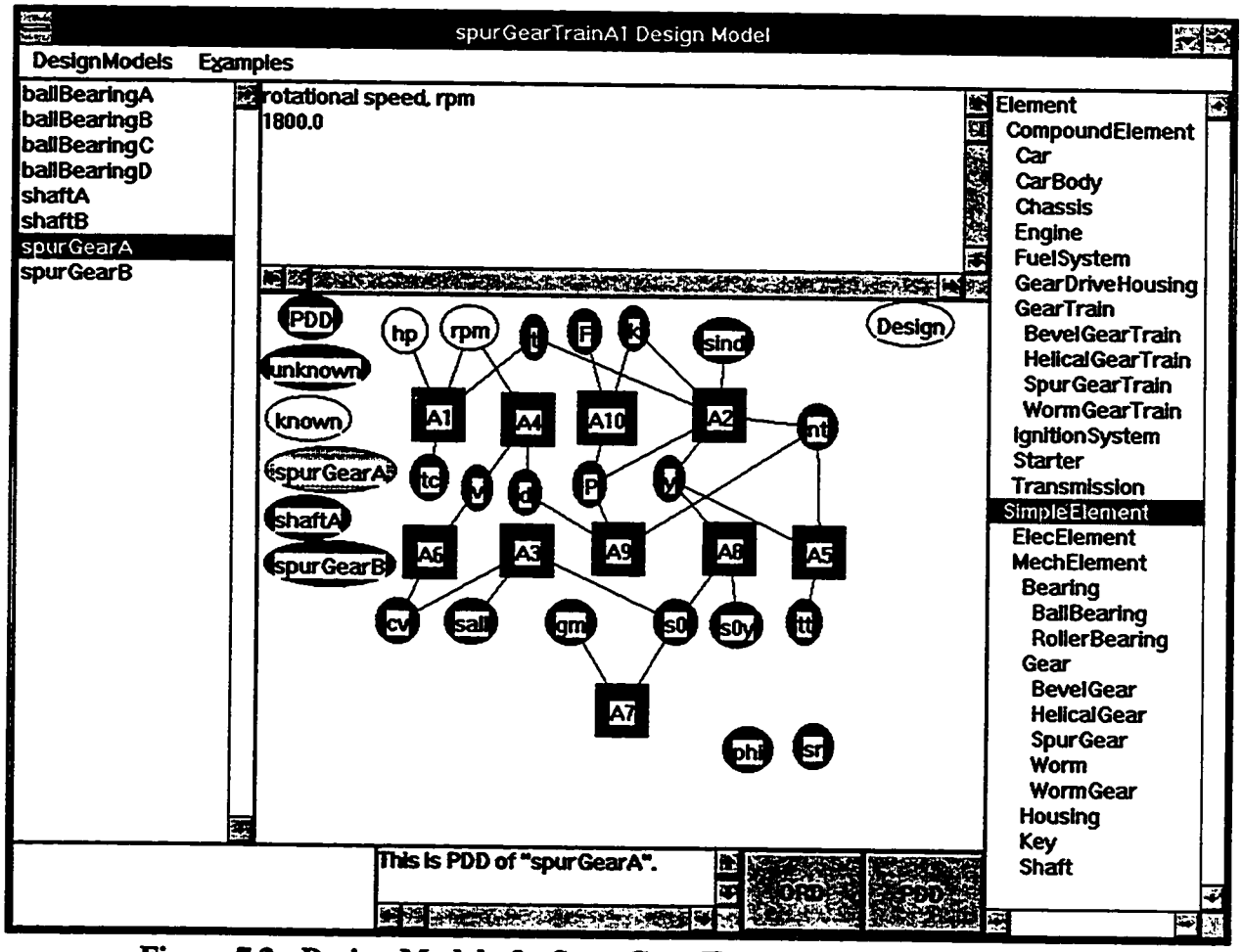


Figure 7.3 : Design Model of a Spur Gear Train with a Spur Gear PDD

on them. The name of an algorithm consists of letter 'A' and a number. The variables are represented by ellipses, with the variable name on them. Red color (dark gray in figure) means unknown variables and yellow color (light gray in the figure) means known variables. Variables are connected

only to algorithms, and algorithms are connected only to variables. There are some unknown variables whose values are related to other parts variables, like the number of the teeth (nt) in spurGearA which is related to spurGearB. These variables are called interface variables. Their color will be different, but variables which are related to the same part have the same color and this color is represented by an ellipse in the left of the graph-pane, with the related part name on it.

In the beginning of the part design, the designer will enter the input parameters values. In the gear train example, the input parameters are the horsepower (hp) and angular speed (rpm). The designer can pass these input parameters values from the input part (shaftA) to other connected parts by choosing 'Power Transfer' item in the PDD menu. This option in the menu will cause the design engine to calculate the power transmitted (hp) and the angular speed (rpm) of all the parts, one at a time.

In each part design, the designer has two options, to pass design control to the design engine which will process the design of that part automatically, or guide the design process himself. These two options are always available to the designer through the part design process. Hence, the designer can pass the design process control to the design engine, or take over the design process control from the design engine, in any stage of the part design. In the automatic design process, the design engine will show the algorithm name which is in process on the message-pane; hence the designer can interrupt the automatic design process at any stage that he desires.

The design engine will check the algorithm which is selected by the designer to be executed. This check will include the number of unknown variables. The number of unknown variables is equal to the number of variables with the color other than yellow, that includes the interface variables which are not calculated yet, but their color in the pdd is not red. If there are no unknowns,

the design engine will give the related message to the designer and, after his acknowledgment, it will wait for another selection by the designer. When there are more than one unknown, the design engine will ask the designer to enter the values of some of the variables, so the algorithm can be executed. When there is only one unknown, the design engine will directly execute the algorithm. The algorithms are written in a way that it does not matter which variable is unknown. The unknown variable will be recognized within the algorithm, then its value will be calculated according to the values of the other variables and their relations.

The designer can change the value of the variables by pointing to the variable and double-clicking the mouse button on it in the graph-pane. The design engine will show the current value before changing it. The designer can also modify the value of the variables by selecting the component in the list-pane and editing it. The list-pane has three states: components, algorithms and the parameters (variables) state. In the parameters and the algorithms states, a list of parameters or algorithms name will be appear in the list-pane. The designer can absorb the description of the parameters and algorithms in the edit-pane by selecting their name in the list-pane.

There are some variables in the pdd which are not connected to any algorithm. These variables either are interface parameters and connected to other parts (e.g., **sr**, speed ratio in gears' pdd which are related to the meshed gear), or will be used in other calculations of the system like pressure angle, **phi**, in the gear train which is used for calculation of the exchange forces between meshed gears.

7.5.2 General Gear Design

In Figure 7.3 which shows the PDD of a spur gear, the Lewis equation for the calculation of the gear teeth strength is used to design the gear. It is assumed that the designer already entered the input parameters (hp and rpm) in the input shaft, and then by using 'Power Transfer' item in the menu the same parameters of the spur gear are also calculated. The algorithms and the variables which are in this pdd are common in all subclasses of the Gear class; hence they are implemented in the Gear class and inherited by its subclasses. Therefore, the design process of this pdd can be assumed as a general gear design, then in other type of gears other necessary variables and algorithms will be added.

The designer can start the gear design by executing algorithm A1 which calculates torque t = 63,030x hp/rpm, then follows execution of the second algorithm (A2) which calculates the induced stress (s_{in}) on the gear teeth. The designer should assign values to some of its variables because there are more than one unknown variables. Usually k (= F/p) which is between 3 to 4. For ordinary services, this can be assumed to be equal to 3.5 (an average value). The number of teeth (nt) and the diametral pitch (P) could have some trial values, but the Lewis form factor (y) should be obtained from algorithm A5 by knowing the tooth type (tt) and the number of the teeth (nt). After obtaining the Lewis form factor, the induced stress (s_{in}) can be calculated from algorithm A2. The calculated induced stress should be less than the allowable stress (s_{all}) on gear teeth. In order to calculate the allowable stress, first the base stress (s_o) will be obtained from algorithm A7 by knowing the gear material (gm), then from algorithm A9 the gear diameter (D), and from algorithm A4, the pitch-line velocity (V) will be calculated according to the angular speed (rpm) and gear diameter (D) values. The algorithm A6 will calculate velocity factor (C_v) and from that using

algorithm A3, allowable stress will be calculated. The designer will compare the values of induced and allowable stresses, and will modify the trial value of the diametral pitch (P) or the face width of the gear accordingly, or it could be done by the design engine by choosing 'Check design' item in menu.

There is a variable in the gears' pdd called s0y which is the multiplication of the base stress (s_o) and the Lewis form factor (y). When the material of the meshed gears is not the same, calculating this variable for both meshed gears, the designer can determine which gear is weaker, and then start designing the weaker gear. Having completed the design, the interface variables of the meshed gear can be calculated according to the weaker gear variables' value. For example, in a pair of meshed gears, the number of teeth (nt), diametral pitch (P), gear diameter (D), and the face width (F) of the stronger gear can be calculated according to the values of the same variables of the weaker gear and speed ratio (sr) of the meshed gears. At the end of each part design, the designer can compare the variables values with standard values, if any are available.

In order to process the other components' design, the design engine needs to know the forces which apply on them. The forces are caused due to the power transmission. The gear pair is the force link between the shafts. The exchange forces between the meshed gears will be transferred through the shafts to the bearings which support the shafts.

7.5.3 Gear to Gear Forces

The next components which should be designed in the gear train are shafts and bearings, but before starting their design, the designer should obtain forces which apply on them. The first step in force analysis is the calculation of the forces which exchange between the gears. In order to

analyze the forces on the gears, the designer should select the 'gearToGearForces' item from the menu. As was mentioned, the design engine will ask the designer to enter a value for any unknown variables which are related to the gear force calculation, for example, the pressure angle of the gears teeth which will be obtained from the gears teeth type.

These forces are calculated in the 'primeProcess:' method of the GearToGear class. Only one gear to gear connection is defined, and there is no subclass for the GearToGear class, which, in fact, is not even required. That is because only the force calculation in gear connections are different. In order to solve this problem, the design engine will use the general equations for force and torque calculation which was derived in Chapter 5, and are as follows:

$$\begin{cases} R_{x,t} = -R_{x,h}\cos\alpha\cos\beta + R_{y,h}\sin\beta - R_{z,h}\sin\alpha \\ R_{y,t} = +R_{x,h}\cos\alpha\sin\beta + R_{y,h}\cos\beta \\ R_{z,t} = -R_{x,h}\sin\alpha\cos\beta + R_{z,h}\cos\alpha \end{cases}$$

$$\begin{cases} M_{x,t} = -M_{x,h}\left(\dfrac{D_t\cos\psi_t}{D_h\cos\psi_h}\right) \\ M_{y,t} = M_{y,h} = 0 \\ M_{z,t} = -\dfrac{D_t}{D_h}\left(M_{z,h}\cos\alpha\cos\beta + D_h R_{y,h}\sin\beta + M_{x,h}\sin\alpha\right) \end{cases}$$

where subscript **t** stands for the tail and subscript **h** for the head component of the connection object, and : R_x , R_y and R_z are the three force components in three dimensions.

M_x , M_y and M_z are the three torque components in three dimensions.

D_h and D_t are the head and the tail gear pitch diameters.

α is the angle between two non-parallel and non-intersected shafts (Figure 5.9), like crossed helical or worm gears.

β is the angle between two intersected shafts (Figure 5.10), such as bevel gears.

α and β are assumed for different type of gears, and at least one of them will be zero in the design process of a single type of a gear pair. In the design of the gears with parallel axes, both angles will be zero.

After this step, the forces and the moments which applies from gears to the shafts will be calculated by selecting the 'shaftToGearForces' in the menu. The next step in the force analysis is to obtain the forces on the supporting bearings. This process will be done by the selection of the 'shaftToBearingForces' item in the menu.

7.5.4 Shaft Minimum Partial Length

Forces analysis on different parts of the shaft are more complicated than other force analyses because the action point of each force should be known. In the beginning of the shaft design, the information does not exist because the locations of the mounted components on the shaft are unknown. In order to obtain the location of these components, the design engine needs to know the partial length of the shaft for each mounted component. As discussed in Chapter 5, the partial length of the shaft depends on the face width of the mounted component and some other information like angle between the axes of the gears. In the first iteration of the design process, due to the lack of these data, the design engine will assume some initial values for the variables which are related to the force analysis of the shaft. For example, for the face width of the bearings the average of one inch and the distance between the components half an inch will be considered. There is a method in each

element class called 'partialLen' which calculates the minimum partial length of the shaft for that mounted component according to the theory discussed in Chapter 5. Hence, the location of the shaft steps and total length of the shaft will be obtained using initial values for the partial lengths. Then, the forces on the bearings can be estimated.

7.5.5 Bearing Design

With initial forces for the bearings, the designer may select each bearing pdd and process its design. Bearings are compound elements which consist of several simple elements (e.g., balls). The aim of the gear train design in this section is not to design all simple elements of the bearing, but to choose the right bearing from a selection of standard bearings by knowing its loads, rpm, and so on. Hence, bearing design is not complicated compared to a gear or shaft design. Figure 7.4 shows a ball bearing pdd; there are only seven algorithms in its pdd. The angular speed (rpm) of the bearing will be calculated in 'Power Transfer' process, then during the bearing force analysis the thrust load (T) and the radial load (R) of the bearing will be obtained.

Having assigned some value to the variables, the design process of the bearings can be continued. Algorithm A2 will obtain the thrust factor (F_t) according to thrust and radial loads. Algorithm A3 will assign value to the speed factor (F_s) according to the angular the speed value. In order to obtain a value for variable Q from Algorithm A4, the designer should select a bearing type. After deciding life time (L) of the bearing, rated radial load will be calculated by using Algorithm A1, then Algorithm A5 will obtain the bearing number (BN) from the stored table according to the rated radial load. Having the bearing number, Algorithms A6 and A7 will obtain the inner diameter (R_i) and the face width of the bearing (F).

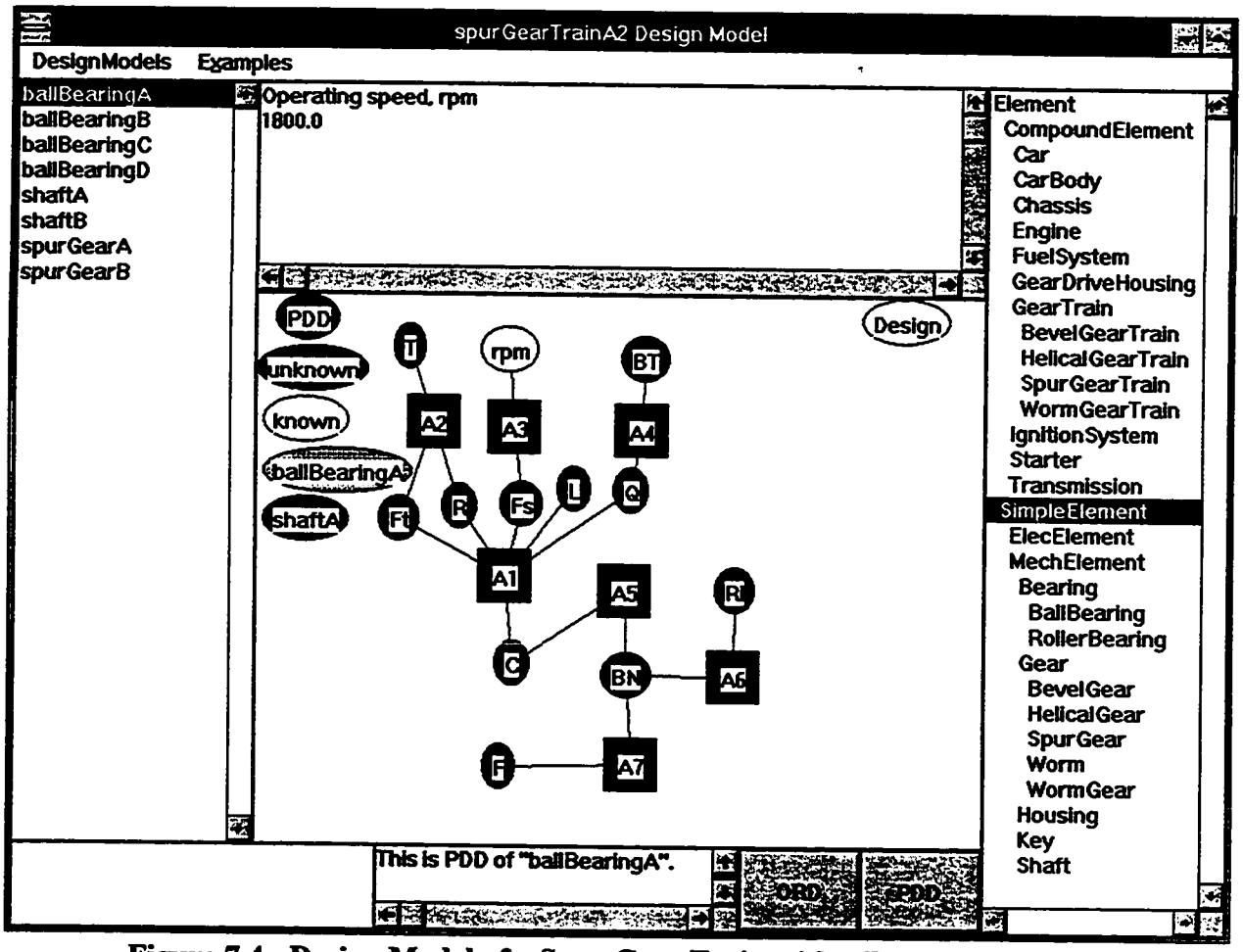


Figure 7.4 : Design Model of a Spur Gear Train with a Ball Bearing PDD

Having the real values for the face width of the bearings, the designer can run the second iteration of the 'shaftToBearingForces'. But before the second iteration, the shaft should be unloaded. That means the shaft variables should be the same as before the first iteration. The designer can unload a shaft by selecting the 'unload' item from menu. Then, the location of the shaft steps, with new values of the partial lengths will be obtained for the second iteration. At this point, the real location of the steps on the shaft and whether the step is up or down will be known. Other specifications of the steps like fillet radius are unknown.

7.5.6 Shaft Design

A shaft is the base for each collection of components mounted on it. Therefore, the coordinates of mounted components will be related to their shaft. As mentioned, the axes of the rotational components are very important, and to simplify the design of such components, the rotational axes of these kind of components are assumed as their 'x' axes. The direction of a shaft's axis is important because the designer should know the start and the end of the shaft for entering the order of the components mounted on that very shaft. So, for finding a direction for a already created shaft, rules are made to be followed by the CAID system and the designer. At the beginning, the general coordinates of the gear train will be considered. If a shaft's rotational axis is parallel to the 'x' axis of the gear train, the shaft axis direction will be the same direction as the gear train 'x' axis. If they are not parallel, the direction of the 'x' axis of the shaft will be assumed in a way that the angle between two 'x' axes to be less than 90 degrees. In special case when these axes are perpendicular, the direction of the 'x' axis of the shaft will be assume in a way that by rotating the shaft axis 90 degrees in counter-clockwise, it will lie on the 'x' axis of the gear train.

The 'y' axis of the rotational components will be considered according to the connection in the process of design. The direction of this axis will be chosen according to the same rules as 'x' axis. The 'z' axis and its direction will be considered according to right-handed coordinate system.

By knowing the direction of the 'x' axis of the shaft, the designer can store the information related to the length of the shaft. The mounted components on the shaft will be stored in an instance variable called 'mountedComponents' which is an ordered collection. These components will be stored in this instance variable in order, according to their positions on the shaft from starting to the end of the shaft. Any other important point on the shaft (e.g., step) will be stored on an instance

variable called 'nodes' in the order of their position on the shaft. Forces and moments applying on the shaft will be stored in an instance variable called 'loads'. Nodes and loads are two sorted collections of the shaft object.

As mentioned, shafts could be stepped before or after mounted components. For this reason, the pdd of the stepped shaft will depend on the number of mounted components and whether the step is before or after the mounted component. The designer can give the order of the mounted components in the connection mode. It is assumed by the CAID system that the first connected component to the shaft is the first mounted component from the left of the shaft, and the order of the connection of the components to the shaft will show the mounting order of the components on the shaft from left to right. If a designer did not follow this order in the connection mode, the correct order could still be given before starting the shaft design. The simplest stepped shaft has at least three mounted components, two bearings and one gear. Hence, the design of stepped shaft is better to be done automatically. This automatic design process is divided to several stages. As mentioned, at first the exchange forces between the meshed gears will be calculated, then the forces which are applied from the gears on the shafts will be obtained. If the gears are connected to the shafts by keys, their forces which apply on the shaft will also be obtained. Finally the forces of the supporting bearings will be calculated. The designer has control between all these stages but the process of each stage will be done by the design engine.

Having all the forces on the shaft calculated, the designer selects a shaft and designs it. There are two stages left, shaft moments calculation and shaft diameters calculation. The design engine will ask the designer to enter the value of the unknown variables, whenever their needed. For example, the design engine will need the axes angle (alpha or beta) or helix angle for the calculation of the

exchange forces.

7.5.6.1 Shaft Moments Calculations

The next stage is to analyze the shaft moments in different sections. The shaft moment calculation can be done by selecting the 'Calculate shaft moments' item in the list-pane menu. This selection will evoke the 'calculateMoments' method in Shaft class. If the selected component is not a shaft, the design engine will issue an error message. In order to implement segments of a shaft's moment diagram, a class called Segment is defined. The instances of this class simulates the linear segments of a moment diagram as in Figure 6.10. These instances have three instance variables named 'initialValue', 'slope' and 'leftDistance'. Their names imply their purpose. There are three different moments on each section of a shaft, two bending moments in 'y' and 'z' direction and a torsional torque which can be assumed in the 'x' direction. Hence, there will be three moment diagrams for each shaft. There are instance variables in each shaft object named momentX, momentY and momentZ, which represent three moment diagrams of the shaft in three coordinate directions. These instance variables are sorted collections. That means their elements, which are Segment instances, will be sorted according to their distance from the shaft's left side (leftDistance value).

As mentioned, there are two important variable in each shaft object named 'loads' and 'nodes'. Both of them are sorted collections. The first one, loads, stores elements like forces and moments on the shafts which are subclasses of the LoadNode class. It is assumed that these loads apply on the shaft in the middle of the component face width. The second one, nodes, stores the important points on the shaft, like steps and keyways which are subclasses of the ShaftNodes class.

The elements of these instance variables are sorted by element locations on the shaft from the shaft's left hand side. This information will help the design engine to calculate the moments and the force diagram of the shaft. In order to build up these moment diagrams, the moment segments are calculated from the left to the right of the shaft. The important points on the shaft such as step, keyway, forces and moments action points, and etc., which are stored in 'nodes' and 'loads' variables affect the moment diagram. Each one of the nodes will be a starting point for a moment segment; hence the left distance of the segment will be equal to the left distance of the node. The slope of the segment will be calculated according to the force applying in the section of the shaft in that node. The initial value of the segment will also be calculated according to the moment applying in the section of the shaft in that node and final value of the previous segment. The maximum moment will happen in the beginning or the end of the moment segments, because the moment segments are assumed to be linear. Hence, the maximum moment is in the nodes.

7.5.6.2 Shaft Diameters Calculations

The next stage is to calculate the diameters of the stepped shaft in each segment according to the moment diagrams. In order to calculate the diameter of the shaft the designer will select the 'Calculate shaft diameter' from list-pane menu. This will evoke the 'calculateDiameter' method in the shaft class. The selected component should be a shaft and its moment diagrams should already be calculated.

The design engine will calculate different stresses (S_{Rx} , S_{Mx} , S_{My} and S_{Mz}) in the sections where there are load on the shaft, or diameter changes. Because the shaft is assumed to be stepped and may have keyways, the process of the shaft diameter calculation will be applied to different parts

of the shaft, which has different section shapes. The location of these sections can be obtained from 'loads' and 'nodes' variables elements. For each divided portion of the shaft the maximum stress will be calculated according to the theory which was analyzed in Chapter 5.

The stress concentrated factors are considered in the fillets of the shaft steps. The stress concentration factors are related to the shaft diameters, before and after the step, and the fillet radius. There will be two iterations in this stage. In the first iteration, the values of the concentration factors (K_t's) will be assumed as one.

When calculating the diameter of a shaft portion, the diameter of the next portion is not calculated yet. That is why the system should assume an initial value for the next portion of the shaft diameter. This will be done by assuming a minimum diameter and minimum step size for each portion and step of the shaft. These minimums will be calculated in 'minStep' and 'minDiameter' methods of Shaft class.

7.6 Keyway in Shaft Design

As mentioned, in the case when there is a keyway on the shaft (Figure 5.8), the system considered the inner diameter in the maximum stress calculation. The maximum shear stress in the section A-A, which is in the risk of failure, will also be checked. This check will be done in 'checkKeyway: failureOf:' method of the Shaft class.

7.7 Numerical Example

The spur gear is the most general gear compared to the other types of gears because the variables and algorithms, which are used in spur gear design, are also valid in the design of other gear types. In Section 7.5, the design process of a spur gear, stepped shaft and bearing, using their pdds, are discussed. In this section, as a numerical example, the spur gear train in Figure 7.5 will be analyzed numerically.

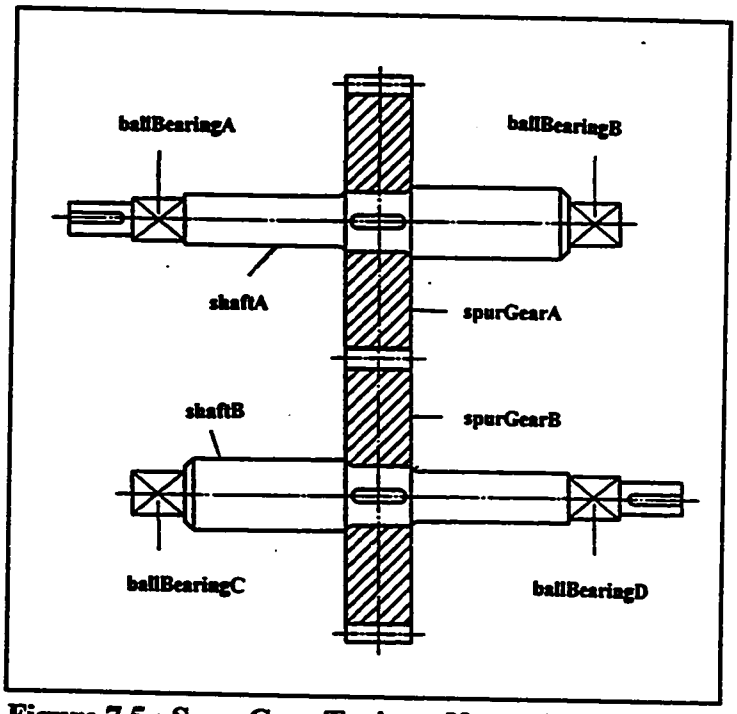


Figure 7.5 : Spur Gear Train as Numerical Example

Supposedly the gear train in hand has the following specifications:

horse power = 15.19 hp,
input angular speed = 1200 rpm,
speed ratio = 2.5 : 1

In order to start the gear train design, the values of the input parameters will be assigned to the input shaft (shaftA) as follows:

hp = 15.19 horsepower,

rpm = 1200 rpm,

selecting 'Power Transfer' from the menu will calculate the same variables for all components of

the gear train.

7.7.1 The Spur Gear Design

The designer may continue the design process by selecting one of the spur gears, and bringing up its pdd in graph-pane by pointing to the PDD button. Before going through the complete design of the selected spur gear, it is better to check the '**soy**' (= $s_o \times y$) variable of both gears to see which one is weaker. To calculate this variable in each gear, the design engine needs to know teeth type (tt), number of teeth (nt), and gears material (gm) for both spur gears. The value of these variable are chosen as follows:

spurGearA:

teeth type (tt) = 20° full-depth,

number of teeth (nt) = 12,

gear material (gm) = SAE 1030,

from Algorithm A7 the basic stress (s_o) for spurGearA will be

s_o = 20000 psi,

and from Algorithm A5 the Lewis form factor (y) for spurGearA will be

y = 0.078,

then from Algorithm A8 soy variable for spurGearA will be :

soy = 1560.0

For spurGearB the teeth type (tt) and the number of teeth are related to spurGearA variables, and the gear material can be chosen as followed:

spurGearB:

teeth type (tt) = 20° full-depth,

number of teeth (nt) = 30,

gear material (gm) = SAE 1030,

from Algorithm A7 the basic stress (s_o) for spurGearB will be

s_o = 20000 psi,

and from Algorithm A5 the Lewis form factor (y) for spurGearB will be

y = 0.114,

then from Algorithm A8 soy variable for spurGearB will be

soy = 2280.0

Concluding from the comparison of 'soy' in both gears (2280 > 1560), spurGearA is weaker than spurGearB. The designer will follow the design process through the pdd of spurGearA. If the designer takes care of the order of the algorithms which are to be executed, their order and results will be as follows:

Algorithm A1 => torque (t) = 797.85 lb-in,

assigning:

k (average value of 3 to 4) = 3.5,

P (diametral pitch, trial) = 10,

Algorithm A10 => F = 1.10 in.,

Algorithm A2 => s_{ind} = 49402.70 psi,

Algorithm A9 => D = 1.2 in.,

Algorithm A4 => V = 376.8 fpm,

Algorithm A6 => $C_v = 0.6143$,

Algorithm A3 => $s_{all} = 12285.01$ psi,

Because 49402.70 psi (= s_{ind}) >12285.01 psi (= s_{all}), some variable values should be changed to reduce the induced stress. The induced stress can be reduced by decreasing the initial value of diametral pitch (P) or increasing the value of the face width (F) of the spur gear. These two variables are related by coefficient **k**, initialized to an average value of 3.5. Increasing the value of this coefficient to its maximum, 4, will not reduce the induced stress to be less than the allowable stress value (s_{all}). Hence the initial value of diametral pitch should be changed according to the stress ratio as follows:

$$\text{new diametral pitch, P} = \text{old diametral pitch} \times \sqrt[3]{\text{stress ratio } (s_{all}/s_{ind})}$$

$$= 10 \times \sqrt[3]{12285.01/49402.70} = 6.29 \approx 6$$

This reasoning and these calculations can be done by the design engine if the designer selects 'auto design' item in the graph-pane menu, after completing the first iteration of the spur gear design. The second iteration of the spurGearA design will be as follows:

having:

$t = 797.85$ lb-in.,

$s_o = 20000$ psi,

$nt = 12$,

$y = 0.078$,

$k = 3.5$,

$P = 6$,

the execution order of the algorithms and their results are:

Algorithm A10 => F = 1.83 in.,

Algorithm A2 => s_{ind} = 10670.98 psi,

Algorithm A9 => D = 2.0 in.,

Algorithm A4 => V = 628.0 fpm,

Algorithm A6 => C_v = 0.4886,

Algorithm A3 => s_{all} = 9771.98 psi,

The induced stress is now comparable to the allowable stress, but it is still greater. In order to reduce the induced stress, increasing the face width (F) may be enough.

new face width, F = old face width × stress ratio (s_{ind}/s_{all})

= 1.83 × 10670.98 / 9771.98 = 2.0

Hence, k = F × P / π = 2.0 × 6 / π = 3.82 is acceptable. The third iteration of the spur gear design will be as follows:

Algorithm A2 => (s_{ind} =) 9771.98 psi = 9771.98 psi (= s_{all}).

Now returning back to the spurGearB design, there are some interface variables related to spurGearA. The design engine can calculate these variables according to related variables values of spurGearA. These variables and their values are as follows:

P = 6,

F = 2.0 in.,

k = 3.82,

nt = 30,

tt = 20° full depth,

D = 5.0 in.,

Although it is not necessary to analyze the stress of spurGearB, being stronger than spurGearA, as a test the designer may calculate the stresses in this gear too.

Algorithm A1 => torque (t) = -1994.64 lb-in,

Algorithm A2 => s_{ind} = 6686.08 psi,

Algorithm A9 => D = 5.0 in.,

Algorithm A4 => V = 628.0 fpm,

Algorithm A6 => C_v = 0.4886,

Algorithm A3 => s_{all} = 9771.98 psi,

As it was mentioned, spurGearB is stronger than spurGearA and

(s_{all} =) 9771.98 psi > 6686.08 psi (= s_{ind}).

These results will be compared with predesigned Boston gears (page A14 from reference [44], Appendix D) as follows:

	spurGearA	Boston gear
gear type :	full depth 20	full depth 20
gear material :	steel (SAE 1030)	steel
base stress (s_o) :	20000 psi	20000 psi
diametral pitch :	6	6
face width :	2.0	2.
rpm :	1200	1200
number of teeth :	12	12
hp :	15.19	15.19
torque :	797.85 lb-in.	798 lb-in.

and,

	spurGearB	Boston gear
gear type :	full depth 20	full depth 20
gear material :	steel (SAE 1030)	steel
base stress (s_o) :	20000 psi	20000 psi
diametral pitch :	6	6
face width :	2.	2.
rpm :	480	480
number of teeth :	30	30
hp :	15.19	21.63
torque :	1994.64 lb-in.	2993.2 lb-in.

As is obvious from the first table, comparing SpurGearA and Boston gear, SpurGearA is acceptable since its torque is almost the same as Boston gears' torque. In the second table, the speed ratio, torque and horsepower values (480, 21.63 and 2993.2) are interpolated from the same table (page A14 from reference [44], Appendix D). The interpolated values of the torque and horsepower are not exactly the same as SpurGearB's torque and horsepower values since SpurGearA was designed while SpurGearB was selected according to speed ratio.

7.7.2 Forces Analysis

As mentioned, in order to process the design of the other components, the design engine needs to know the forces which apply on them. The forces causes because of the power transmission. The gear pair is the force link between the shafts. The exchange forces between the meshed gears will be transferred through shafts to bearings which supporting the shafts.

The next components which should be designed in the gear train are shafts and bearings, but before staring their design, the designer should obtain forces which apply on them. First step in force analysis is the calculation of the forces which exchange between the gears. In order to analysis the

forces on the gears the designer should select the 'gearToGearForces' item from menu. As was mentioned, the design engine will ask the designer to enter the value for any unknown variable which is related to gears forces calculation. In this case, because there is only one pair of spur gears, the angle between the axes will be zero. The pressure angle of the gears teeth will be obtained from gears teeth type, which in this case is 20°. The forces between the gears will be stored in the 'headForces' and 'tailForces' variables of the gear connection object, and are as follows:

Rx ==> 0.0 lb,

Ry ==> -290.40 lb,

Rz ==> -797.85 lb,

After this step, the forces and the moments, which apply from gears to the shafts, will be calculated by selecting the 'shaftToGearForces' in the menu. The forces on the shafts are as follows:

For shaftA:

Rx ==> 0.0 lb,
Ry ==> -290.40 lb,
Rz ==> -797.85 lb,
Mx ==> -797.85 lb-in.,
My ==> 0.0
Mz ==> 0.0

For shaftB:

Rx ==> 0.0 lb,
Ry ==> -290.40 lb,
Rz ==> -797.85 lb,
Mx ==> -1994.64 lb-in.,
My ==> 0.0
Mz ==> 0.0

Force analysis on different parts of the shaft are more complicated than other force analysis since the action point of each force are initially unknown, which is due to a lack of exact data in exactly locating the shaft mounted components. In order to obtain the location of these components, the design engine needs to know the partial length of the shaft for each mounted component. As it was discussed in Chapter 5, the partial length of the shaft depends on the face width of the mounted component and some other information, like the angle between the axes of the gears. In the first iteration of the design process, because of the lack of data, the design engine will assume some initial values for the variables which are related to the force analysis of the shaft.

The next step in force analysis is to obtain the forces on the supporting bearings. This process will be done by the selection of the 'shaftToBearingForces' item in the menu. For stepped ShaftA, the steps will be assumed according to Figure 7.5. These steps are located after ballBearingA, spurGearA and before ballBearingB. Hence, with initial values for the partial lengths, the initial total length of the shaft is 4.80 inches and the forces on the bearings are as follows:

$R_{x,B1} = R_{x,B2} = 0.0$ lb,

$R_{y,B1} = R_{y,B2} = 145.1$ lb,

$R_{z,B1} = R_{z,B2} = 398.9$ lb,

7.7.3 Bearings Design

The thrust on the bearings are zero and the total radial forces on the bearings are:

$R_1 = R_2 = 424.53$ lb,

With these bearing forces, the designer may select each bearing pdd and process its design. The process for both ballBearingA and B, with some of the variables initialized, will be as follows:

ballBearingA & B:

T (trust) = 0

R (radial load) = 424.53

BT (Bearing Type) = NDur300 (Type 3000 New Departure),

L (Desired life) = 20000 hrs,

Algorithm A2 => $F_t = 1.01$,

Algorithm A3 => $F_s = 0.9554$,

Algorithm A4 => Q = 2280,

Algorithm A1 => C = 772.36 lb,

Algorithm A5 => BN = 3304.

Algorithm A6 => $R_i = 0.7874$ in.,

Algorithm A7 => F = 0.5906 in.

These values were selected, by the CAID, from stored ballbearing dimensions which are also provided in tables from reference [21]. Having the real values for the face width of the bearings, the designer can run the second iteration of the 'shaftToBearingForces'. But before the second iteration, the shaft should be unloaded. Meaning, the shaft variables should be the same as before the first iteration. The designer can unload a shaft by selecting 'unload' item from the menu. Then, the location of the shaft steps with new values of the partial lengths will be obtained for the second iteration. At this point, the real location of the steps on the shaft and which step is up or down are known. Other specifications of the steps like fillet radius are unknown. Total length of the shaft becomes 3.98 inches and the forces on the bearings are as follows:

$R_1 = R_2 = 424.53$ lb,

the forces applied on the bearings are not changed. Hence, the bearings types and proportions will be the same.

7.7.4 The Shaft Design

The next step is to analyze the shaft moments in different sections. In this step, the designer will select 'CalculateShaftMoments' from the menu or algorithm 'A1' from shaft's pdd. There are two important variables in each shaft object named 'loads' and 'nodes'. Both of them are sorted collections. 'loads' stores the forces and moments on the shafts. Assuming these loads apply on the shaft in the middle of the component face width. 'nodes' stores the important points on the shaft like steps and keyways. The elements of these instance variables are sorted by element location on the shaft from the shaft's left side. This information will help the design engine to calculate the moments and force diagram of the shaft. These diagrams will be stored as segments in sorted collection variables called 'forceX', 'momentX', 'momentY' and 'momentZ'. The maximum moment of each segment will be stored in variable 'momentMax' which is also a sorted collection.

Figure 7.6 shows the forces applied on shaftA. There is no force on 'x' direction. The gears and bearing forces in 'y' and 'z' directions are applied in the middle of their face width. The moment diagrams of the shaft can be derived by using the flexural and torsional loading analysis on shafts from any references about the mechanics of materials (e.g., [24]). For M_x or torque, the CAID system will ask whether shaft is an input or output shaft, and from which end of the shaft is the torque transmitted into or out of the shaft. According to Figure 7.5, it is assumed that shaftA is the input

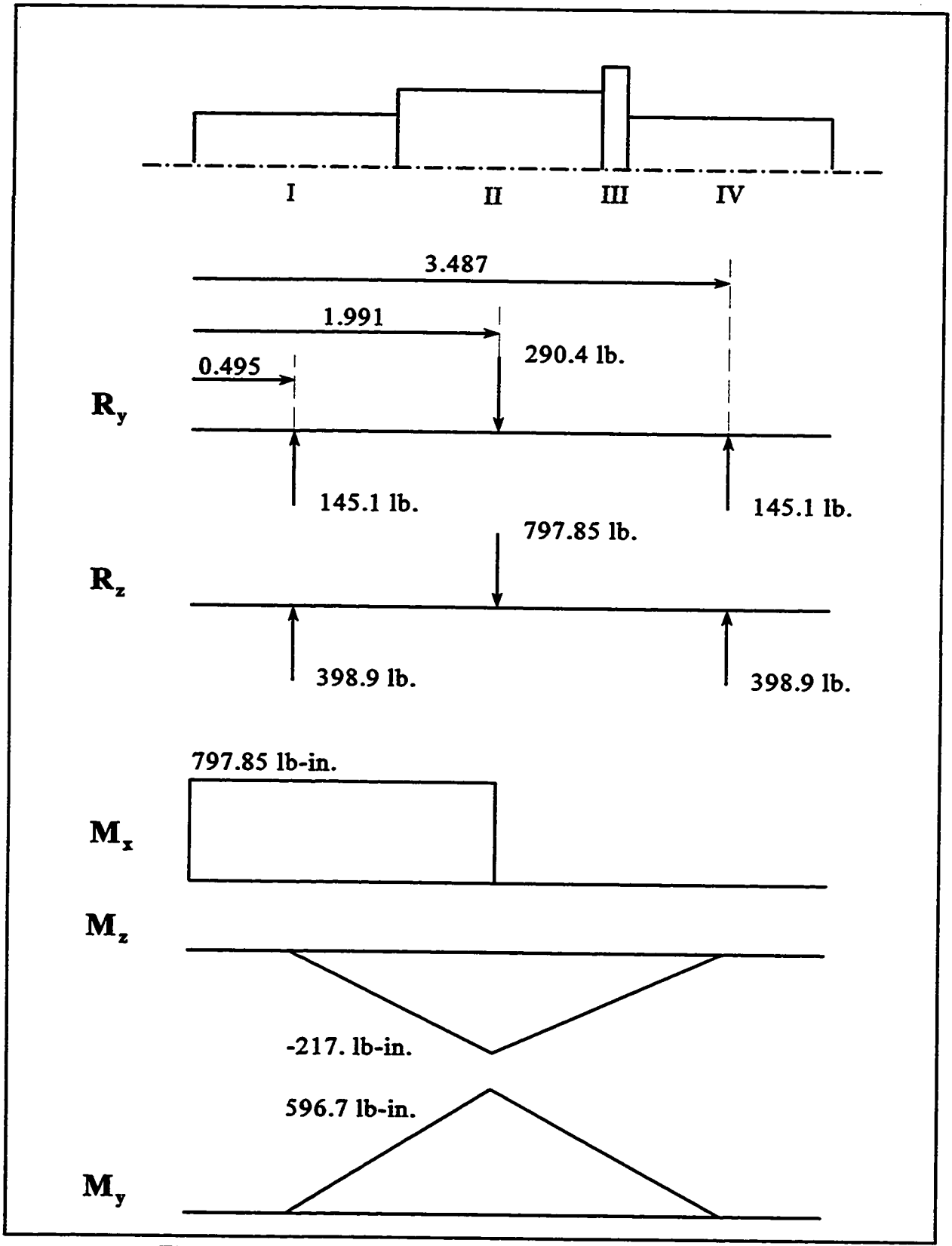


Figure 7.6 : Forces and Moments Applied on Stepped ShaftA

shaft and the torque is transmitted to the shaft from its left end. Hence, the moment diagrams (i.e.,M_x M_y M_z), will be as Figure 7.6. The real output of the CAID system are shown in Figures 7.7, 7.8 and 7.9.

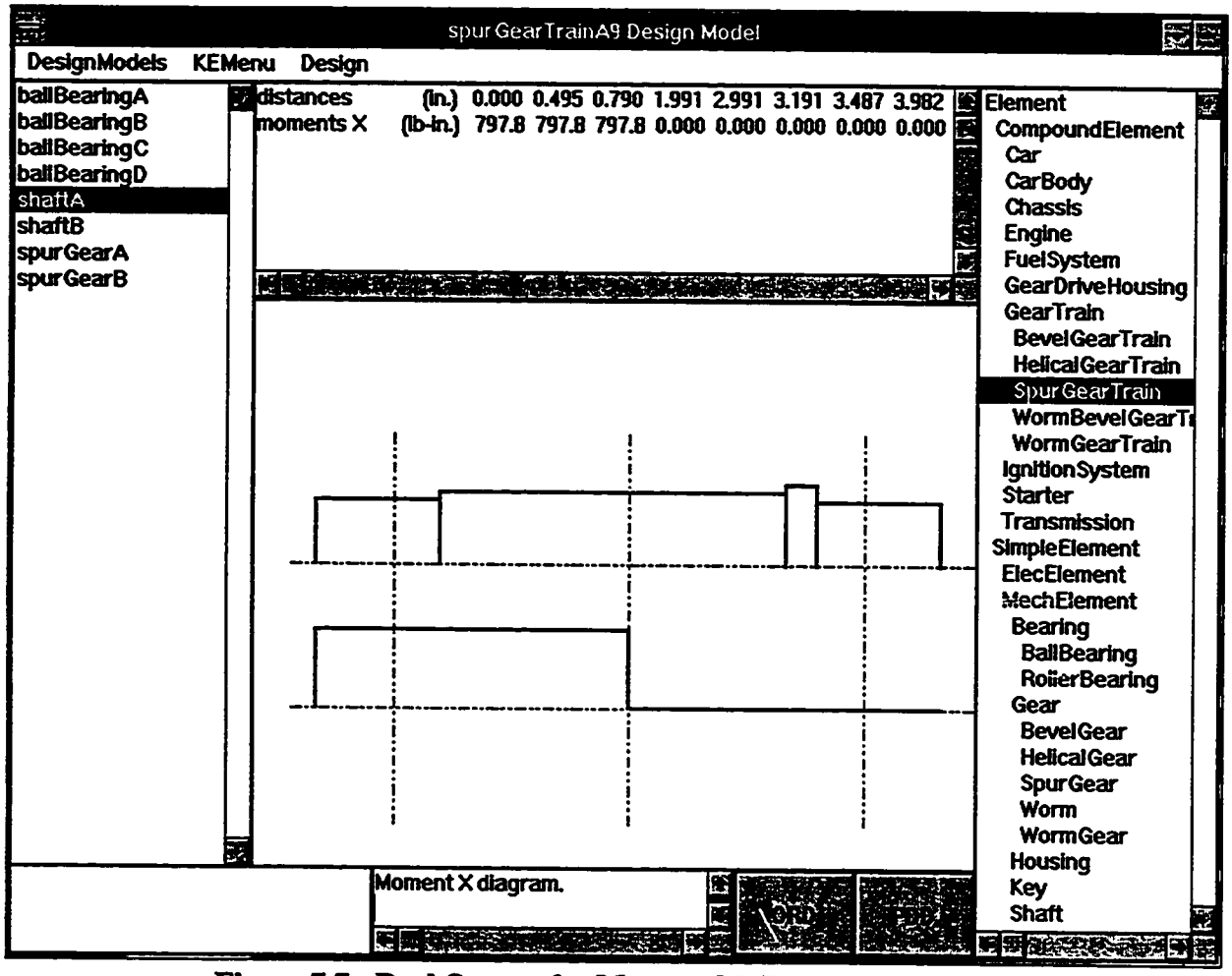


Figure 7.7 : Real Output for Moment M_x Diagram of ShaftA

The next stage is to calculate the diameters of the stepped shaftA in each segment according to these diagram data. In order to calculate these diameters, the design engine will calculate different stresses (S_{Rx} , S_{Mx} , S_{My} and S_{Mz}) in the sections where there are loads on the shaft, or the diameter changes. The location of these sections can be obtain from 'loads' and 'nodes' variable elements.

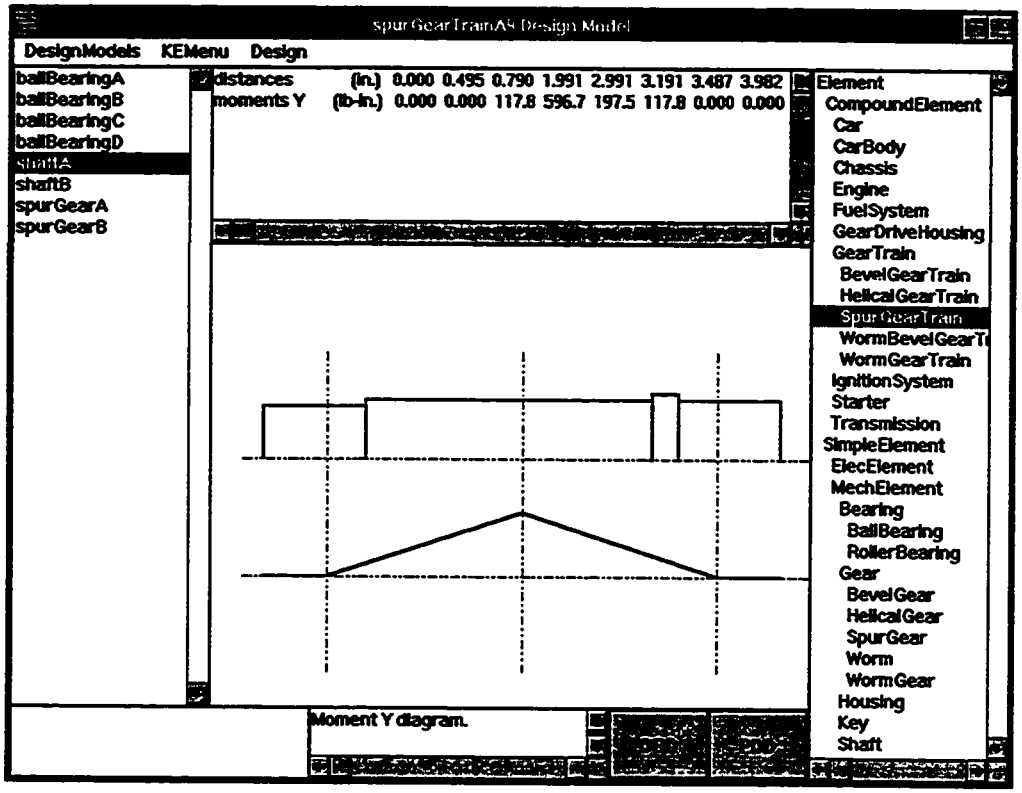


Figure 7.8 : Real Output for Moment M_y Diagram of ShaftA

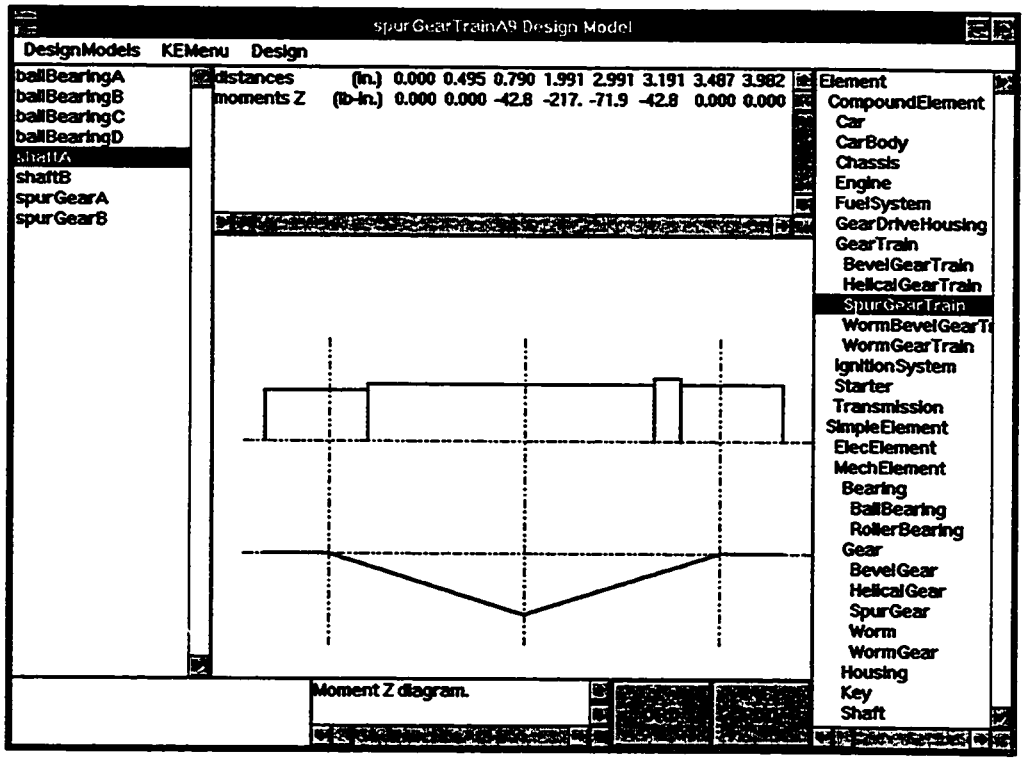


Figure 7.9 : Real Output for Moment M_z Diagram of ShaftA

There will be two iterations in this stage. In the first iteration, the values of the concentration factors (K_t's) will be assumed as one. The diameter of the shaft in different sections should be calculated according to the allowable stress of the shaft, as follows:

shaft material = SAE1020 => S_{all} = 13750 psi,

The diameter in each can be calculated by using equation 5.49

$$D_{sh} = \sqrt[3]{\frac{16}{\pi} \cdot \frac{K_t \sqrt{T^2 + M^2}}{S_{all}}}$$

where D_{sh} is minimum shaft diameter,

K_t is stress concentration factor,

T (torque) = M_x ,

$M^2 = (M_y)^2 + (M_z)^2$.

In the first iteration, due to lack of the diameter values, K_t can not be calculated. Hence, K_t is assumed to be 1 (no stress concentration).

section I :

$M_{x(max)}$ = 797.85 lb-in.

$M_{y(max)}$ = 117.8 lb-in.

$M_{z(max)}$ = -42.8 lb-in.

D_{sh} = 0.6688 in.

section II :

$M_{x(max)}$ = 797.85 lb-in.

$M_{y(max)}$ = 596.7 lb-in.

$M_{z(max)}$ = -217. lb-in.

D_{sh} = 0.7228 in.

section III :

$M_{x(max)} = 0.0$ lb-in.

$M_{y(max)} = 197.5$ lb-in.

$M_{z(max)} = -71.$ lb-in.

$D_{sh} = 0.4268$ in.

section IV :

$M_{x(max)} = 0.0$ lb-in.

$M_{y(max)} = 117.8$ lb-in.

$M_{z(max)} = -42.8$ lb-in.

$D_{sh} = 0.3594$ in..

The inside radius of the Ballbearings is 0.7874 in., which is greater than radiuses of the shaft in sections I and IV, where bearings are supposed to be mounted. Because the calculated value for the diameters are a minimum value, they can be increased to fit the bearings inside diameter. Hence, the diameters of the shaft in section I and IV will be:

$D_{sh} = 2 \times 0.7874 = 1.5748.$

The diameters in section II and III should be greater than diameters in the previous sections (i.e., I and II). Assuming the minimum step size to be 0.1 in., the diameters in section II and III will be 1.7748 and 1.9748 in., respectively.

In the second iteration, stress concentration will be considered as in equations 5.50 and 5.51.

$$S_{R_x} = K_{t1}\frac{R_x}{A} \qquad S_{M_x} = K_{t2}\frac{(d/2)M_x}{J}$$

$$S_{M_y} = K_{t3}\frac{(d/2)M_y}{I} \qquad S_{M_z} = K_{t3}\frac{(d/2)M_z}{I}$$

Stress concentration factors (K_t's) depend on D_{sh}/d_{sh} and r/d_{sh} where D_{sh} and d_{sh} are the shaft diameters at both side of the step and **r** is the fillet radius. The fillet radius is assumed to be 0.2 in. Hence, the normal and shear stress of the shaft can be calculated as follows:

$$\sigma = \sqrt{S_{R_x}^2 + S_{M_y}^2 + S_{M_z}^2} \quad and \quad \tau = S_{M_x}$$

$$then \quad \sqrt{\left(\frac{\sigma}{2}\right)^2 + \tau^2} = S_{max} < S_{all}$$

section I :

$M_{x(max)}$ = 797.85 lb-in.
$M_{y(max)}$ = 117.8 lb-in.
$M_{z(max)}$ = -42.8 lb-in.
D_{sh}/d_{sh} = 1.7748/1.5748 = 1.127,
r/d_{sh} = 0.2/1.5748 = 0.127,
K_{t2} (for torsion) = 1.3,
K_{t3} (for bending) = 1.6,
I = 0.3019 & J = 0.6038,
σ = 523.02 psi & τ = 1352.60 psi,
S_{max} = 1377.65 psi.

section II :

$M_{x(max)}$ = 797.85 lb-in.
$M_{y(max)}$ = 596.7 lb-in.
$M_{z(max)}$ = -217. lb-in.
D_{sh}/d_{sh} = 1.9748/1.7748 = 1.113,
r/d_{sh} = 0.2/1.7748 = 0.113,
K_{t2} (for torsion) = 1.2,
K_{t3} (for bending) = 1.6,
I = 0.4870 & J = 0.9741,
σ = 1851.14 psi & τ = 872.20 psi,
S_{max} = 1271.77 psi.

section III :

$M_{x(max)} = 0.0$ lb-in.

$M_{y(max)} = 197.5$ lb-in.

$M_{z(max)} = -71.$ lb-in.

$D_{sh}/d_{sh} = 1.9748/1.5748 = 1.254,$

$r/d_{sh} = 0.2/1.5748 = 0.127,,$

K_{t2} (for torsion) = 1.3,

K_{t3} (for bending) = 1.6,

$I = 0.7466$ & $J = 1.493,$

$\sigma = 444.1$ psi & $\tau = 0.0$ psi,

$S_{max} = 222.05$ psi.

section IV :

$M_{x(max)} = 0.0$ lb-in.

$M_{y(max)} = 117.8$ lb-in.

$M_{z(max)} = -42.8$ lb-in.

No step.

K_{t2} (for torsion) = 1,

K_{t3} (for bending) = 1,

$I = 0.3019$ & $J = 0.6038,$

$\sigma = 326.89$ psi & $\tau = 0.0$ psi,

$S_{max} = 163.45$ psi.

As can be seen, the maximum stress on all sections of the shaft are less than the allowable stress of the shaft. The real output of the CAID system for the final diameters are shown in Figure 7.10.

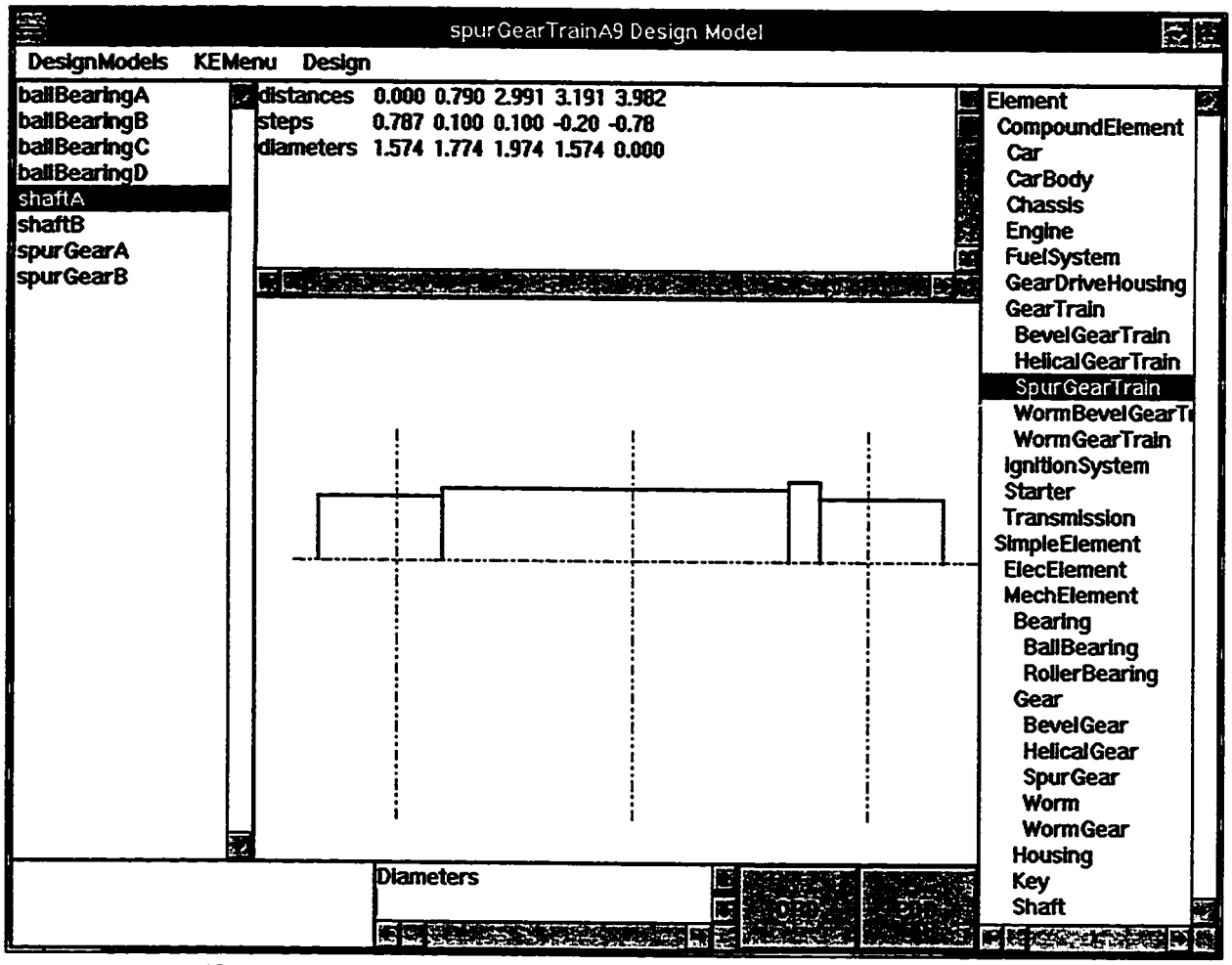


Figure 7.10 Real Output for Diameters of the Stepped ShaftA

Chapter 8

Conclusions

8.1 Summary

This project was initiated because previous researchers were unable to concurrently achieve automation, integration and creativity in design process. The CAID system has many advantages which are not achieved in the other mechanical engineering design expert systems developed until now. Previous research aims were to design a fully automatic system. Usually this automation costs in the loss of creativity of the design. These systems usually accept the specifications of the design as the system input, and pass the final design results as the system output. In the best case, all feasible design alternatives are presented for a final choice-stage where users can pick the best proposal according to their own evaluation criteria. The fully automated system user has no control on the design process.

In this research a new environment is suggested to organize a computer automated and integrated design (CAID) system. This thesis introduced CAID and its philosophy, then described how it can make use of the power of the latest computer technology. This system has been implemented using the object-oriented approach.

The CAID is organized in such a manner that the expert knowledge stored in it could be easily retrieved by the junior designer. The experience and the knowledge of the expert are placed in the bases. Multiple design models, in the CAID system, can be processed simultaneously because the expert knowledge is broken into elements which can be used by different design models. This way the system is more flexible and extendable, which gives more freedom and creativity to the

designer.

The CAID system grows and changes the technical knowledge by having a senior designer contributes his design knowledge to the system. This knowledge is organized in the CAID system in a modular building block fashion, and a junior designer who uses the system for design purpose has access to all this knowledge. The designer is able to control the design process, and process more than one design case simultaneously. The designer is able to switch between these design cases, and store completed design cases and use them as a starting point for future designs.

The confidentiality of the design process is very important in design offices. With CAID, each design office can store their own special knowledge of the design within the system, and no other office, even the provider of the CAID system cannot, without authorization, access that special knowledge have to know anything about that special knowledge.

The accomplishment of this research achieves a significant advance in the area of automated design and presents a real contribution to the field. Graphical interface for simple component part design (PDD) and compound component object relations (ORD) were developed for the first time to be used in an interactive user interface. These diagrams can be manipulated automatically whenever a designer desires. The idea of the CAID system is novel and takes a much more complete advantage of the design capabilities of computers.

To verify the validity of the system, gear train design knowledge is implemented in the system and tested. To implement the knowledge of different gear train components, the gear and the shaft design process are generalized. There are no recorded attempts, so far, to generalize gear and shaft design in a way applicable to all gears and shafts. Allowing this research to be one of the pioneers in generalizing design processes, especially as applied to gear analysis.

Comparing the CAID system to existing software like intelligent spreadsheets, one may deduce that each spreadsheet can play the role of a PDD while a notebook, consisting of several spreadsheets, can play the role of an ORD. By studying the CAID system, it will be obvious that it offers significant advantages over the spreadsheet approach. For example, the relation between two mechanical components can be easily defined by using Connection subclasses, but in a notebook the relation between two spreadsheets can be defined only by relating some of their parameters to each other. This process is required of the spreadsheet user every time a connection is created. However, in CAID, the process will be done automatically according to the type of related components requiring much less effort and time and with better results.

As mentioned, using the object-oriented approach, implementing CAID, gives the system the ability to provide generic elements. These generic elements are represented by names so the element can be created simply by pointing to its name, allowing it to be easily added to the layout of the design in hand. A hierarchy of elements which the user can point to and a copy (instance) of the pointed element will be created automatically. New elements (simple/compound) can also be added to this hierarchy. The knowledge engineer can modify the ORD and PDD of those elements. These features cause the CAID system to be remarkably superior to the former spreadsheets.

8.2 Future Work

There are future development steps which can expand the CAID system, and make it more advanced and user friendly. In this section, some of these steps will be recommended.

A program can be created to make the algorithms of the pdd automatically asking for the necessary information from the designer. This program can have a parsing section like a computer

programming language compiler to convert the mathematical relations to Smalltalk codes. This program can be an interface between the CAID system and the knowledge engineer. It would make the algorithms in standard form which will make the system more consistent.

Another program can be implemented to make the pdd of the elements automatically. This program needs a graphical interface between the system and knowledge engineer. In the present system, this task is performing up to a certain level, but requires improvement.

As mentioned, at the time of the system components creation, there are element classes which do not have instances. It will be a good idea if the user can create a general form of element, if, in the beginning of the design, the element type suitable for the design is not known. For example, in a gear train design, the designer can create an instance of a gear, then during the design process, the proper type of the gear, after having more information about the conditions that will apply.

There are other improvement options to make CAID more user friendly. For example, the type of the connection can be represented on the connection line in graph-pane, and the user can change or edit it by pointing on it.

In the part design diagram, there were different types of variables. The designer could assign values to the unknown or interface variables, or change already assigned values, which may cause some inconsistency. In order to solve this problem, it would be better to divide the variables in two categories: the variables which have values that can be changed by the designer, and the variables which can not be changed and which CAID would take care of.

It would be a good idea if the design engine checks the consistency of all the algorithms related to a modified variable. Hence, the design engine should modify at least one other variable to satisfy the relation between the variables. In order to decide which variables should be modified,

the design engine could put aside fixed variables (variables which can not be modified by the designer), then choose the variables in the order given by knowledge engineer in advance.

It would be better to show the real stepped shaft (in one dimension), so that the designer can decide the order of the mounted components and whether the step is before the mounted component or after it.

It would be beneficial if the CAID system would be connected to a Computer-Aided Design (CAD) system, to work in parallel. In this case, by any progress in design process, the designer could see the progress on the CAD system in real form.

By implementing the CAID system on a Visual Age Smalltalk Server, the system could be used for processing complicated designs. Several designers could work on the same complicated design simultaneously. If one designer adds some class or method to the system, other designers could use these additions. A design manager would control CAID on the server. To design a complicated design like a car, he or she could pass the design of each subsystem to a designer through local networking. When the design manager selects a subsystem to be designed, the design model of the subsystem would be opened on the computer chosen by the manager. Since the CAID system on the server would control all of these design models on different computers at the same time. The design manager would choose the best copy of each subsystem and complete the design system (the car). The integration of the CAID system in this manner could present a design vehicle of enormous assistance to the engineering community.

References

1. Fahim, A. "Philosophy of a Computer Automated and Integrated Design (CAID) System", Mechanical Engeering Dept., University of Ottawa 1989.

2. Shariat Panahi, M. " A Computational Approach to Conceptual Design of Mechanical Systems" Ph.D. Dissertation Mech. Eng. Dept. University of Alberta, August 1995.

3. Koller, R. "Ein Algorithmisch-Physikalisch Orientierte Konstructionsmethodik" Z. VDI 115, 1973, (from reference [2]).

4. Ward, A.C. and Seering, W.P. " Quantitative Inference in a Mechanical Design 'Compiler' " Jornal of Mechanical Design Vol. 115, pp. 29-35, March 1993.

5. Chern, J.H. "An Intelligent Knowledge based Approach to Support Engineering Design Practice", Computers in Engineering - Volume One ASME 1991, pp 103-110.

6. Kannapan, S. and Marshek, K.M. "Design Synthetic Reasoning" Mech. Mach. Theory vol. 26, No. 7, pp, 711-739, 1991.

7. Maher, M.L. and Longinos, P. "Development of an Expert System Shell for Engineering Design", Int. J. Appl. Engng Ed. vol. 3, pp 279-286, 1987.

8. Erdman, A.G. and Sandor, G.N. "Mechanism Design Analysis and Synthesis" volume I, Chapter two, Prentice-Hall international, inc. 1991, pp 73-95.

9. Ulrich, K.T. "Intelligent Tools for Mechanical Design", Artificial Intelligence at MIT Expanding Frontiers, Volume 1, 1990, pp 52-69.

10. Finger, S., and Dixon, J.R. "A Review of Research in Mechanical Engineering Design Part I & Part II" Research in Engineering Design, part I, Vol. 1, No.1 & part II, Vol. 1, No. 2, 1989.

11. Dixon, J.R. "New Goals for Engineering Education", ASME, Mechanical Engineering/March 1991, pp 56-62.

12. Daizhong, S. and Forgie, R.J. "Development of an Expert System Approach to an Engineering Design Procedure", Proceedings of the Institution of Mechanical Engineers, Volume II, August 1989, pp 975-990.

13. Soldan, D.L. "Engineering design: What is it?", IEEE Potentials, Volume 7, No. 1, Feb. 1988, pp 6-8.

14. Maher, M.L. "Synthesis and Evaluation of Preliminary Designs", Artificial Intelligence in Design, Vol. 2, 1989.

15. Howe,A.E., Cohen,P.R., Dixon,J.R. and Simmons,M.K., "Dominic: A Domain-Independent Program for Mechanical Engineering Design," Artificial Intelligence, Vol. 1, No. 1, 1986.

16. Dixon, J.R., Howe,A., Cohen,P.R. and Simmons, M.K.,"Dominic I: Progress Towards Domain Independence in Design By Iterative Redesign," Proceedings, ASME 1987 International Computers in Engineering Conference, American Society of Mechanical Engineers, Chicago,Il, July 24-26, 1987.

17. Eastman, C.M., "Recent Developments in Representation in the Science of Design." Proceedings of the 18th Design Automation Conference, ACM, IEEE, June 1981, pp 13-21.

18. Yoshikawa,H., "Automation of Thinking in Design," Computer Applications in Production and Engineering, North-Holland, Amsterdam, 1983, pp 405-417.

19. Fellows, J.W. "All about Computer-Aided Design and Manufacturing", Sigma Technical Press, U.K., 1983.

20. Kardos, G. "Problem Solving with the Fuller-Polya Diagram" Int. J. Appl. Engng Ed. Vol. 1, No.2, pp. 105-111,1985.

21. Black, P.H. and Adams, O.E. "Machine Design" Third Edition McGraw-Hill, 1968.

22. Juvinall, R.C. "Fundamentals of Machine Component Design" McGraw-Hill, 1983.

23. Burr, A.H. and Cheatham, J.B. "Mechanical Analysis and Design" Second Edition Prentice Hall, 1995.

24. Higdon, A., Ohlsen, E.H., Stiles, W.B., Weese, A.J., Riley, W.F. "Mechanics of Materials" John Wiley & Sons 1985.

25. Buckingham, E., "Design Loads on Gear Teeth" the Industrial Press New York, 1931.

26. Roa, M., Wang, Q. and Cha, J. "Integrated Distributed Intelligent Systems in Manufacturing.", Chapter IV, Chapman and Hall, London, UK, 1993.

27. Kannnapan, S. "Design Synthetic Reasoning: A Theoretical Basis and Methodology for Mechanical Design" Ph.D. Dissertation, University of Texas at Austin, Tex. Dec 1989.

28. Kannapan, S. and Marshek, K.M. " An Algebraic and Predicate Logic Approach to Representation and Reasoning in Machine Design" Mech. Mach. Theory vol. 25, No. 3, pp, 335-353, 1990.

29. Wiener, R.S., Pinson, L.J. "An Introduction to Object-Oriented Programming and C++", Addison-Wesiey Publishing Company,1988.

30. Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorensen, W. "Object-Oriented Modeling and Design" Prentice Hall, 1991.

31. Bakker, J.W.,de Roever, W.P., Rozenberg, G. (Eds) "Foundations of Object-Oriented Languages" Lecture Notes in Computer Science, Proceeding of REX Scchool/Workshop, Noordwijkerhout, The Netherlands, Springer-Verlag, May/June 1990.

32. Digitalk "Smalltalk 32-bit Object-Orinted Programming System for Win32, Programming Referece", Digitalk Inc., 1993.

33. Masini, G. and et al "Object Oriented Languages", A.P.I.C. Series Academic Press, 1991

34. Booch, G. "Object Oriented Design with Applications", The Benjamin/Cummings Publishing Company,Inc. 1991.

35. Digitalk "Smalltalk 32-bit Object-Orinted Programming System for Win32, Tutorial", Digitalk Inc., 1992-93.

36. Agha, G. "The Structure and Semantics of Actor Languages", Foundations of Object-Oriented Languages, Lecture Notes in Computer Science, Proceeding of REX Scchool/Workshop, Noordwijkerhout, The Netherlands, Springer-Verlag, May/June 1990, pp 1-59.

37. Salustri, F.A. and Venter, R.D. "An Axiomatic Theory of Engineering Design Information", Engineering with Computers 8, 1992, pp 197-211.

38. Salustri, F.A. and Venter,R.D. "Towards a New Computational Model for Engineering Software Systems" Proceeding of ICED 93, 9th International Conference on Engineering Design, N.F.M.Roozenburg, ed. pp 1359-1368, 1993.

39. America, P. "Designing an Object-Oriented Programming Language with Behavioural Subtyping", Foundations of Object-Oriented Languages, Lecture Notes in Computer Science, Proceeding of REX Scchool/Workshop, Noordwijkerhout, The Netherlands, Springer-Verlag, May/June 1990,pp 60-90.

40. Lalonde, W.R. and Pugh, J.R. "Inside Smalltalk" Prentice Hall 1990.

41. Tello E.R. "Object-Oriented Programming for Artificial Intelligence" Addison-Weseley Publishing Company, Inc. 1989.

42. Gui,J.K. "A Mechanical Design Prototype-Based Reasoner", Laboratory of Information Processing Science, Helsinki University of Technology, Finland, 1991.

43. Streeter, V. L. and Wylie, E.B. "Fluid Mechanics", McGraw-Hill, 1981.

44. Boston Gears Catalog, Quincy, Ma, 1992.

Appendix A

The Object Modeling Technique (OMT)

A.1 Introduction

Object-oriented technology is not just a way of programming, but it is a way of thinking abstractly about a problem using real world concepts. The Object Modeling Technique (OMT) [30] is a comprehensive methodology for object-oriented development. In this methodology there are three models: the Object Model, the Dynamic Model and the Functional Model. The object model describes the objects in the system and their relations. The dynamic model describes the interactions among objects in the system. The functional model describes the data transformations of the system. The OMT uses graphical notations for representing object-oriented concepts. Figures A-1 to A-3 which are from reference [30] show some of these important notations.

A.2 Object Model

Figure A.1 describes the notation for the object model. The symbol for a class is a box with the name of the class in boldface in it. The *attributes* of the class are listed in the second part of the box, and the *operations* (methods in Smalltalk) in the lower third part of the box. An instance of a class (an object) is represented by a rounded box. The class name in parentheses is at the top of the object box in boldface. Object names are in normal font. A *generalization* notation represents the inheritance relation among the classes of a class hierarchy. *Aggregation* is a strong form of

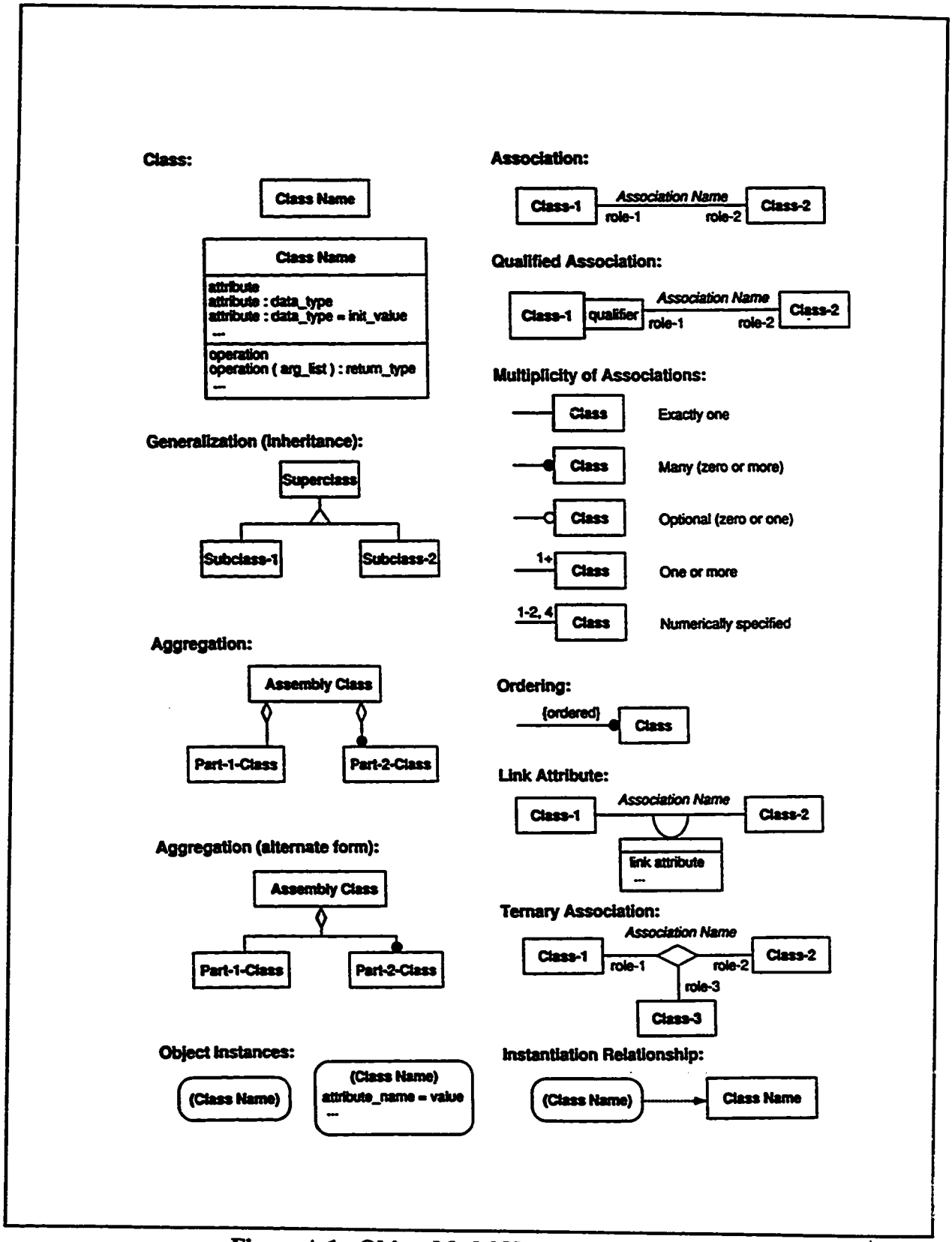


Figure A.1 : Object Model Notation (from [30])

association in which an aggregate object is made of several components. This is called a 'part-of' relation in some other references.

A *link* is a physical or conceptual connection between objects. An *association* describes a group of links. The figure shows a binary and a ternary association. A *role* is one end of an association. A binary association has two roles, each of which may have a role name that uniquely identifies one end of an association. *Multiplicity* specifies how many instances of one class may relate to a single instance of an associated class. The figure shows different case notations for multiplicity. In the case of 'many' in multiplication when the objects are in order, it is indicated by '{ordered}' written next to multiplicity notation.

A link can have attributes just as objects. In many-to-many associations an attribute is the property of the link and can not be attached to either objects. In other types of association it is possible to attach the attributes to objects although it is not preferred. A *qualified* association relates two object classes and a qualifier. The qualifier is a special attribute that reduces the effective multiplicity of an association. Qualification often occurs in real problems, frequently because of the need to supply names.

A.3 Dynamic Model

Figure A.2 describes the notation for the dynamic model. A system can best be understood by first examining its static structure, that is, the structure of its objects and their relationships to each other at a single moment in time. Changes to the objects and their relationships are examined over time. Those aspects of a system that are concerned with time and changes are the dynamic model. The major dynamic modeling concepts are events and states.

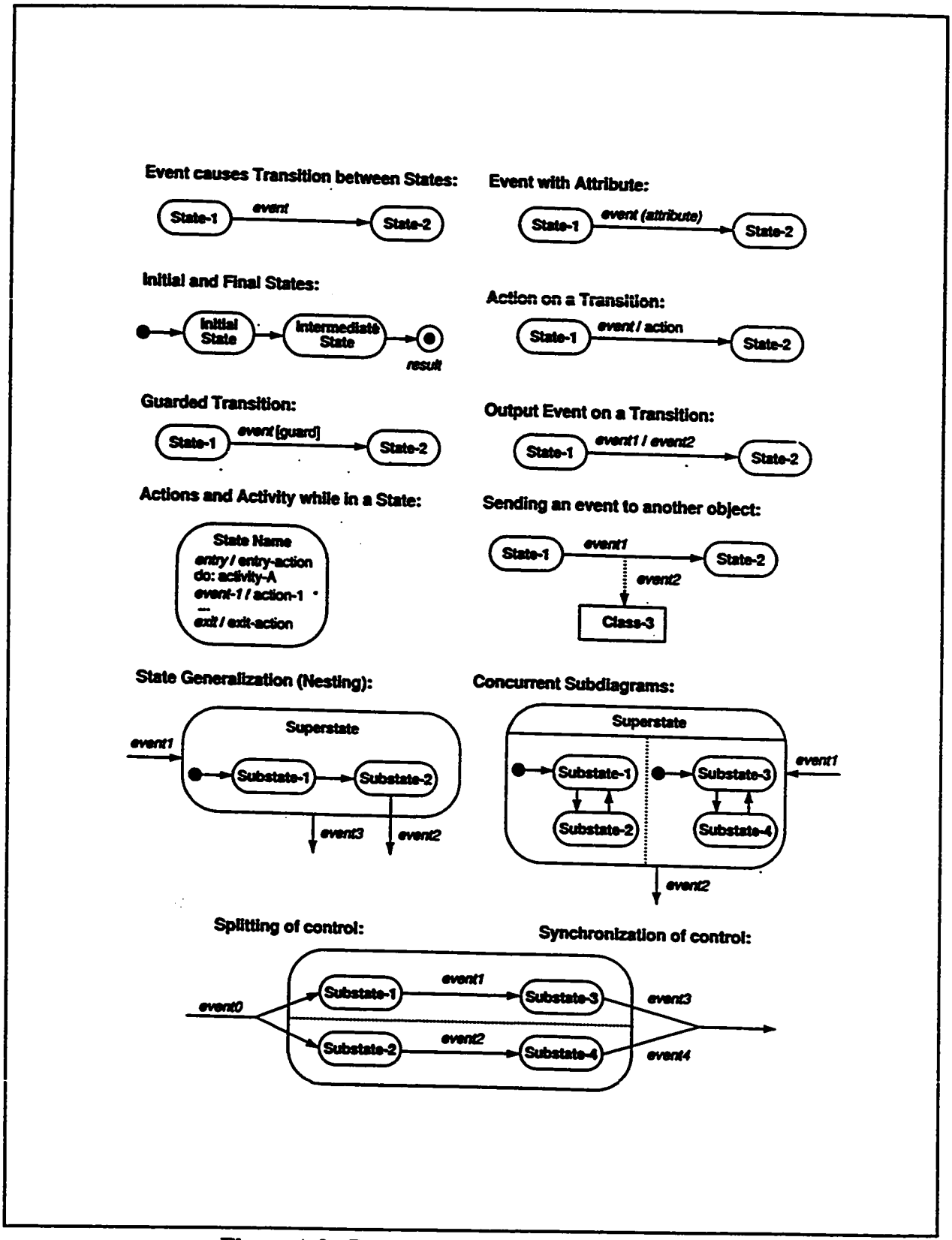


Figure A.2 : Dynamic Model Notation (from [30])

An *event* is something that happens at a point in time. An event is simply an occurrence which has no time, and may cause the state of a system to change. A *state* is an abstraction of the attribute values and links of an object. A state specifies the response of the object to input events. Generalization in a state diagram is equivalent to expanding nested activities. A *state diagram* relates events and states. A dynamic model describes a set of concurrent objects, each with its own state and state diagram. *Concurrency* within the state of a single object arises when the object can be partitioned into subsets of attributes or links, each of which has its own subdiagram. *Synchronization* becomes necessary when one object must perform two (or more) activities concurrently, and the internal steps of the activities are not synchronized.

A.4 Functional Model

Figure A.3 describes the notation for the functional model. The functional model shows how output values in a computation are derived from input values without regard for the order in which the values are computed. A *process* transforms data values. At the implementation stage, a process is generally referred to as a *method.* An *actor* is an active object that drives the data flow graph by producing or consuming values. Actors are attached to the inputs and outputs of a data flow graph. A *data store* is a passive object within a data flow diagram that stores data for later access.

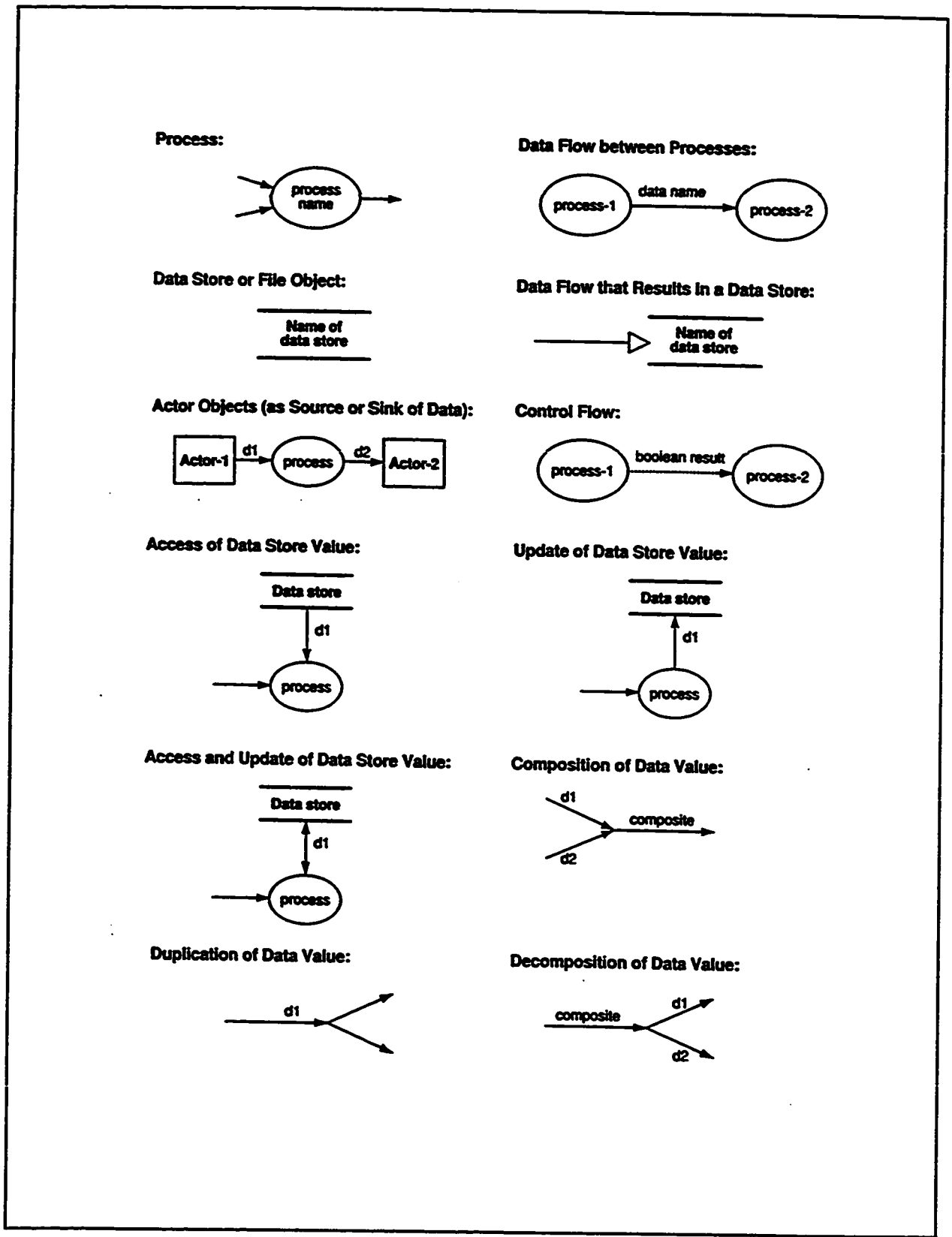


Figure : A.3 Functional Model Notation (from [30])

Appendix B

Gears Terminology

Gear Terminology

The *pitch surface* is the surface of the rolling cylinder that the gear may be considered to be replaced.

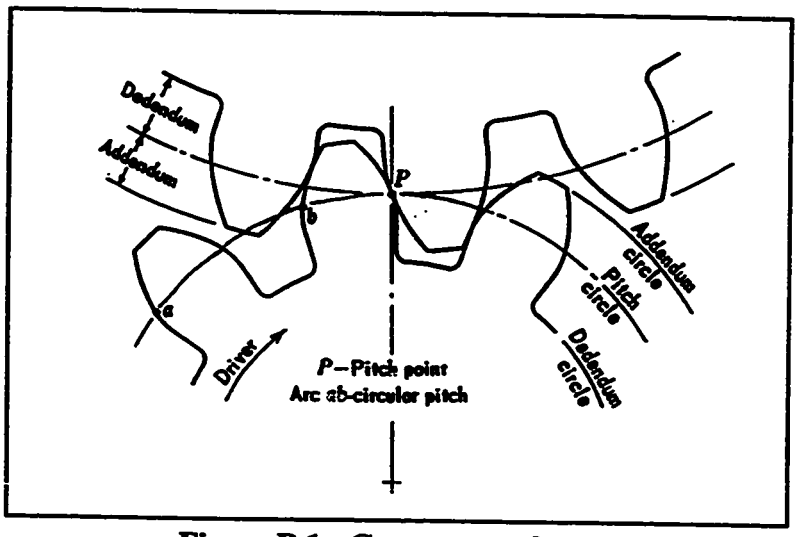


Figure B.1 : Gear nomenclature

The *pitch circle* is a right section of the pitch surface.

The *addendum circle* is the circle bounding the tips of the teeth.

The *dedendum circle* is the circle bounding the bottom of the spaces between the teeth.

The *addendum* is the radial distance between the addendum circle and pitch circle.

The *dedendum* is the radial distance between the pitch circle and the dedendum circle.

Clearance is the difference between the dedendum of one gear and the addendum of the

mating gear.

Backlash is the difference between the tooth space of one gear and tooth thickness of the mating gear measured on the pitch circle.

Circular pitch is the distance from a point on one tooth to the corresponding point on the adjacent tooth measured on the pitch circle. (p in inches).

Diametral pitch is the number of teeth on a gear per inch of its pitch diameter. (P).

Appendix C

Lewis Form Factor (Y)

No. of teeth n	14.5° composite a = 1/P b = 1.157/P	20° full-depth a = 1/P b = 1.157/P	20° stub a = 0.8/P b = 1/P	Radii from AGMA Standard 201.02B 20° a = 1/P b = 1.25/P	25° a = 1/P b = 1.25/P	Full tangent radius* 20° a = 1/P b = 1.25/P	25° a = 1/P b = 1.25/P
10	0.055	0.064	0.088				
11	0.062	0.072	0.093				
12	0.067	0.078	0.099		0.077		0.082
13	0.071	0.083	0.103		0.082		0.088
14	0.075	0.088	0.108		0.086		0.092
15	0.078	0.092	0.111		0.091		0.096
16	0.081	0.094	0.115		0.095		0.100
17	0.084	0.096	0.117		0.099		0.104
18	0.086	0.098	0.120	0.081	0.103	0.093	0.108
19	0.088	0.100	0.123	0.088	0.107	0.095	0.112
20	0.090	0.1021	0.125	0.085	0.111	0.097	0.116
21	0.092	0.104	0.127	0.088	0.116	0.099	0.119
22	0.093	0.105	0.129	0.090	0.120	0.101	0.123
24	0.095	0.107	0.132	0.094	0.128	0.105	0.130
26	0.098	0.110	0.135	0.098	0.136	0.109	0.136
28	0.100	0.112	0.137	0.010	0.130	0.113	0.131
30	0.101	0.114	0.139	0.105	0.133	0.116	0.134
34	0.104	0.118	0.142	0.113	0.138	0.128	0.139
38	0.106	0.122	0.145	0.121	0.143	0.130	0.143
43	0.108	0.126	0.147	0.120	0.147	0.124	0.147
50	0.110	0.130	0.151	0.125	0.152	0.128	0.152
60	0.113	0.134	0.154	0.129	0.156	0.132	0.156
75	0.115	0.138	0.158	0.134	0.161	0.135	0.161
100	0.117	0.142	0.161	0.138	0.166	0.139	0.166
150	0.119	0.146	0.165	0.142	0.169	0.143	0.169
300	0.122	0.150	0.170	0.146	0.175	0.148	0.175
Rack	0.124	0.154	0.175	0.151	0.180	0.154	0.180

* A true radius tangent to both tooth profile and to the root circle.

Appendix D

Boston Spur Gears

6 DIAMETRAL PITCH **STEEL** — **20°** PRESSURE ANGLE — **2"** FACE — REFERENCE PAGE A43.

25 RPM		50 RPM		100 RPM		200 RPM		300 RPM		600 RPM		900 RPM		1200 RPM		1800 RPM		3600 RPM	
H.P.	Torque	H.P.	Torque	H.P.	Torque	H.P.	Torque	H.P.	Torque	H.P.	Torque	H.P.	Torque	H.P.	Torque	H.P.	Torque	H.P.	Torque
.63	1559	1.24	1565	2.38	1502	4.41	1391	6.16	1294	10.20	1072	13.06	915	15.19	798	18.14	635		
.83	2097	1.62	2046	3.10	1951	5.67	1786	7.84	1647	12.70	1334	16.02	1122	18.42	967	21.67	759		
.93	2345	1.81	2284	3.45	2171	6.27	1977	8.64	1814	13.85	1455	17.35	1215	19.85	1043	23.20	812		
1.01	2551	1.97	2480	3.73	2351	6.76	2129	9.26	1945	14.71	1545	18.30	1282	20.85	1095	24.21	848		
1.18	2981	2.29	2889	4.32	2722	7.74	2440	10.52	2210	16.41	1724	20.18	1413	22.79	1197	26.18	917		
1.46	3671	2.81	3541	5.25	3306	9.26	2919	12.44	2613	18.93	1988	22.91	1605	25.61	1345				
1.70	4294	3.27	4122	6.05	3815	10.54	3322	14.00	2941	20.83	2188	24.88	1743	27.56	1448				
1.98	4986	3.78	4763	6.94	4372	11.92	3755	15.66	3241	22.85	2400	26.98	1889						
2.25	5660	4.27	5381	7.77	4899	13.18	4155	17.17	3607	24.60	2584	28.75	2013						

From page A14 reference [44]