Design Descriptions to Support Reasoning about Tolerances

Andrew Keith Robertson

Ph.D.

University of Edinburgh

1992

Abstract

This thesis is concerned with the use of Artificial Intelligence techniques to support human designers. The thesis argues that support for human designers can be improved by adopting an AI-based rather than a geometry-based approach to engineering design. Design Support Systems (DSSs) are proposed as an effective means of delivering this improved support. Representing and reasoning about tolerance statements in design is introduced as a valid area to test these claims.

Tolerance statements describe the allowable variations in the geometry of a designed artefact. Two distinct, but related problems involving the use of tolerance statements in design are tackled, namely: tolerance combination (including the way tolerance distributions combine), and tolerance allocation. The problem of tolerance combination (and distribution) involves determining the necessary consequences of the application of known tolerance statements to one or more designed artefact features. Tolerance allocation concerns the assignment of tolerance statements during the design process. Solutions to this second problem are essential before manufactured instances of designed artefacts can be tested for compliance with design descriptions.

The use of an experimental DSS, the Edinburgh Designer System (EDS), to solve design problems is illustrated. The implementation of techniques to improve the support of tolerance combination and tolerance allocation is described and where possible has been tested using EDS. The way that design is situated within the product creation process is investigated and the derivation of parts list information from an EDS design description is demonstrated. The thesis concludes that the AI-based approach can improve support for human designers, but that further research will be required to demonstrate the effective delivery of this support through DSSs.

Acknowledgements

This work was supported by a Science and Engineering Research Council CASE studentship in collaboration with Lucas Automotive Ltd. (formerly Lucas CAV Ltd.).

My thanks are due to the many people who have helped me over the last six years. In particular, I am indebted to my supervisor Tim Smithers for his guidance, patience and encouragement. I have also benefited from arguments and discussions with my friends at the Department of Artificial Intelligence in Edinburgh, particularly Mike Cameron-Jones and the members of the Edinburgh Designer System team, specifically Karl Millington, Jon Corney, Brian Logan and Dave Corne. In addition, I am grateful to Lucas and the Alvey *Design-to-Product* project for providing an opportunity to test and demonstrate some of the AI-based design ideas presented in this thesis. Finally, thanks to Sophie and M for the unfailing support without which this thesis would not have been completed.

Declaration

This thesis has been composed by myself and describes my own work.

Table of Contents

List of Figures

Chapter 1

Introduction

This thesis is concerned with the use of Artificial Intelligence techniques to support human designers. The thesis argues that support for human designers can be improved by adopting an AI-based rather than a geometry-based approach to engineering design. Design Support Systems (DSSs) are proposed as an effective means of delivering this improved support. Representing and reasoning about tolerance statements in design is presented as a valid area to test these claims.

In this chapter, the motivation and structure of the thesis are described. The motivation section outlines the background to the hypotheses, tests and results presented here. Specifically, the deficiencies of the traditional model underlying the application of computers to design and product creation are discussed. The advantages claimed for an alternative AI-based approach are introduced. The use and manipulation of tolerance statements in design is proposed as a valid area to test these claims.

The chapter closes with a presentation of the thesis structure in the form of a brief summary of each chapter's contents.

1.1 Motivation

The desire to develop computer-based systems which more effectively support the process of product creation motivates this thesis. Underlying this motivation is a dissatisfaction with the way computer-based systems are currently applied and an intellectual curiosity as to whether Artificial Intelligence (AI) techniques can do better.

The CAD/CAM Data Flow Model

Currently, the model of product creation underlying the development of Computer-Aided Design (CAD) and Computer-Aided Manufacture (CAM) systems is based on data flow around a product cycle. This thesis refers to this model as the CAD/CAM data flow model. Figure 1–1 shows a version of this model taken from Groover and Zimmers' CAD/CAM textbook [Groover & Zimmers, 1984].

Chapter 6 discusses the reasons for integrating systems developed for the individual activities shown in Figure 1–1 and suggests why this model is so seductive. Unfortunately, there are a number of major problems with using this model of product creation as a basis for developing computer-based systems to support design. For example:

- Design process capabilities are constrained by the data passed between activities. The development of CAD systems has meant that this communicated data is primarily geometric.

- The model separates the design process from the realisation of design descriptions as designed artefacts. Reasoning about processes outside design, such as production, is not admitted during design engineering.

- The model is frail. For example, there is no indication how problems associated with process planning new design descriptions could be resolved. There

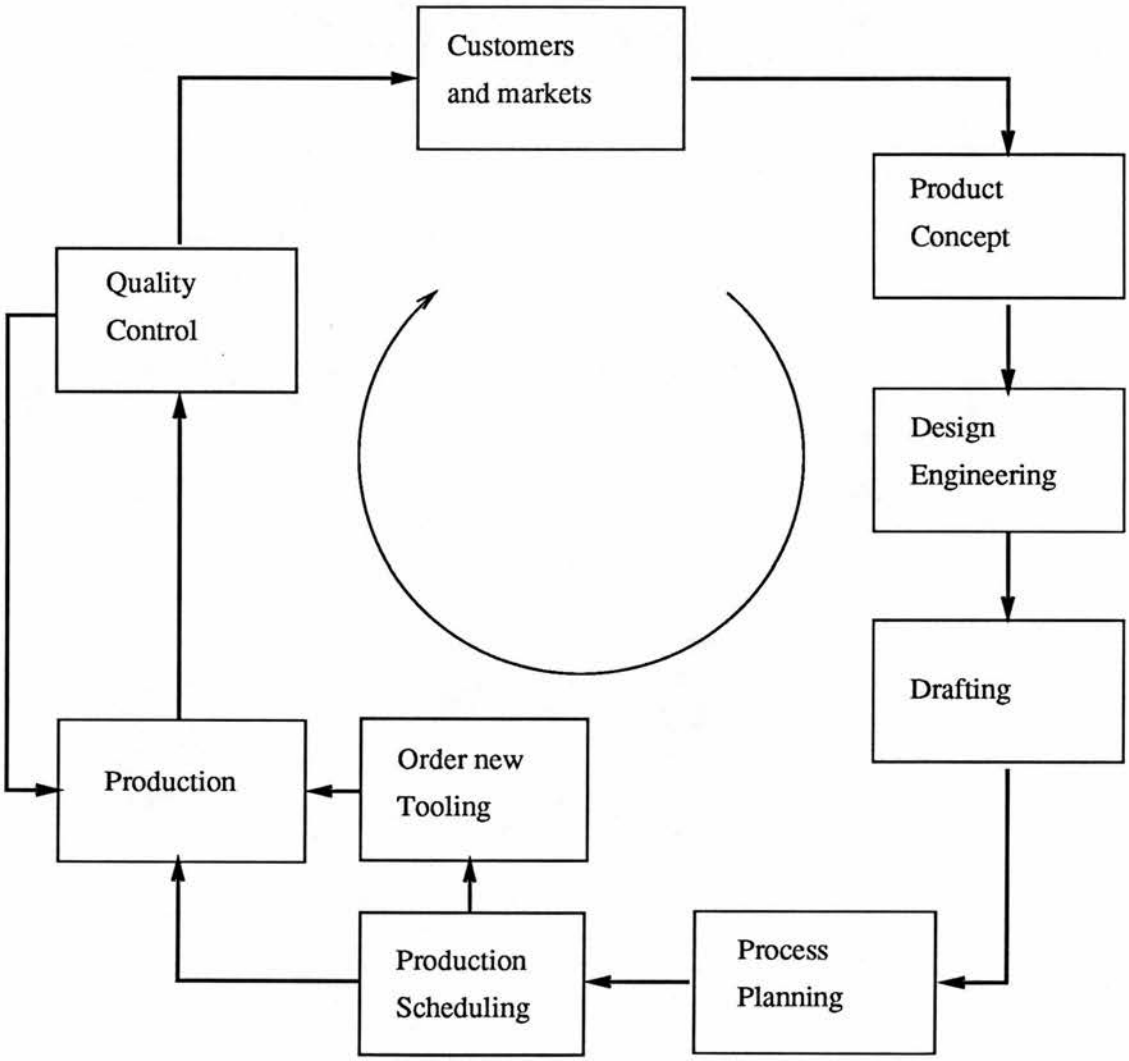


Figure 1–1: The Data Flow Model of Product Creation

is no mechanism for exploring alternative designs and comparing their ease of manufacture: all design outputs result in satisfying customers and markets.

Artificial Intelligence researchers believe that their research programme might offer better tools and techniques with which to support the alternative models of product creation which overcome these deficiencies.

Geometry-based Design

Within the CAD/CAM data flow model the design process creates a design description to be process planned from an original product concept. The description of the design is primarily geometric with additional information necessary for process planning to proceed, for example tolerance statements and material specifications, being added at the stage shown as drafting in Figure 1–1.

The representation of designed artefact geometry in computer-based systems is a research field in its own right. Valid and efficient representations are required for the curves, surfaces and solids of the artefact and these representations may include both topological and geometric data explicitly [Faux & Pratt, 1979]. Beyond the representational issues, CAD system users also require facilities to support the creation, modification and visualisation of their geometric models [Rooney & Steadman, 1987].

In this thesis geometry-based design is used to describe the design process which delivers geometric models of the designed artefact. In geometry-based design, tolerance statements are described as appended to an underlying geometric model based on nominal dimensions. Unlike AI-based design, geometry-based design does not attempt to represent the original product concept or the designer's assumptions and their consequences which underpin the resulting designed artefact geometry.

Artificial Intelligence and Design

Several common elements have emerged from the various attempts to apply AI techniques to understanding, supporting, or automating design processes. These characteristic features are: automated reasoning, knowledge representation, intelligent control, and consistency maintenance [Smithers, 1989]. It is argued that by employing these elements improved models of design and product creation can be supported. For example, knowledge representation research admits the possibility of design descriptions which are not geometry-based, but which reflect the dependencies between the geometric specification and the requirements statement motivating the design process.

The task of applying AI techniques *en masse* to the product creation process is clearly beyond the scope of this thesis. Arguably, this was the original aim of the Alvey *Design-to-Product* project [Smithers, 1985]. The necessity for producing demonstrable results with the potential for commercial exploitation from such grandiose projects undermines the testing of experimental hypotheses [Robertson, 1990]. As a consequence, *Design-to-Product*'s evidence that AI techniques improve the support of product creation is inconclusive. Chapter 6 of this thesis re-visits some of the general issues raised by *Design-to-Product* and the way design is situated within product creation. In contrast to these general issues, the remainder of the experiments presented in this thesis concentrate on a restricted group of design activities. Specifically, this thesis examines whether the AI-based approach improves support for human designers reasoning about tolerance statements.

Experience derived from time spent in a commercial design office and production engineering department (part of the CASE Studentship supporting this research) indicated the variety of uses tolerance statements have. Most commonly, tolerance statements ensure that designed artefacts can be manufactured to function correctly, but they may also describe process capabilities during production, characterise interchangeability in assembly and for maintenance, or determine requirements for inspection equipment. In other words, the use and manipulation of tolerance statements is a pervasive and significant part of the product creation

process. Representing and reasoning about tolerance statements is thus a valid area to test the claims of AI-based design.

Tolerance Terminology

Tolerance statements can describe the allowable variations in the geometry of a designed artefact. International and national standards exist for the form of these tolerance statements [ISO, 1982; BSI, 1984]. Typically, tolerances in the form of upper and lower bounds are given for each linear and angular dimension on a design drawing. This kind of tolerance specification is called +/− tolerancing or conventional tolerancing.

Problems with conventional tolerancing occur as dimensions are not always definable on an instance of a designed artefact because that artefact has been imperfectly formed. Geometric tolerancing and more mathematically formal tolerancing theories address this problem by defining zones relative to perfectly-formed geometry in which features of the designed artefact instance must lie. (A brief description of Requicha's theory of geometric tolerance [Requicha, 1983b] is given in Chapter 2.)

Despite its shortcomings, conventional tolerancing and not geometric tolerancing has been adopted as the basic tolerance representation for the tests and experiments in this thesis. Using conventional tolerancing allowed simple extensions to the experimental Design Support System's constraint syntax to include variational data within design descriptions. By evading the implementation of geometric tolerancing, attention could be focussed on the uses made of tolerance statements in design. In particular, tolerance combination and tolerance allocation have been investigated (see Chapters 4 and 5).

The experimental tolerance representation used by this thesis is described and discussed in Chapter 3 and shown in Figure 1–2 (c). Rather than explicitly represent the upper and lower limits of size as in Figure 1–2 (a) with the tolerance value inferred from the difference between these figures, an explicit representation of the tolerance value is adopted with this value being *equally* and *bilaterally* disposed

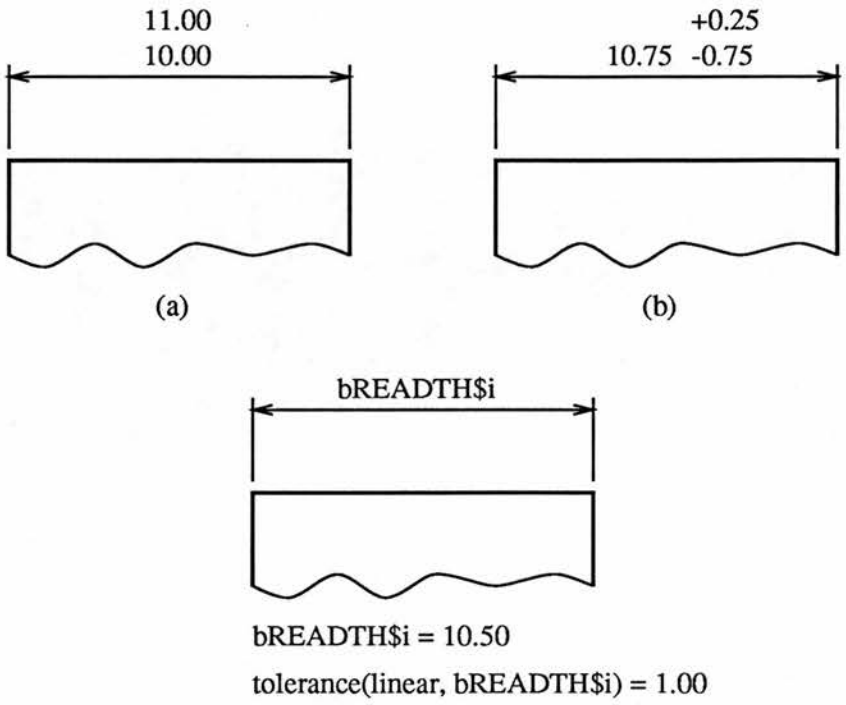


Figure 1–2: Tolerancing Dimensions

about the nominal value. For the Design Support System, the nominal value is the instantiated value of a design parameter such as bREADTH for an instance label such as i and the linear tolerance is a value associated with the parameter instance through a tolerance expression (Figure 1–2 (c)).

This tolerance representation is equivalent to a restricted form of the conventional bilateral tolerancing (shown in Figure 1–2 (b)). As a consequence, the Design Support System does not represent the limiting deviation from the nominal (equal to half the tolerance value) explicitly. Where this is a significant parameter, as in the use of the quality loss function (see Chapter 5), its value must be derived from the tolerance.

1.2 Motivating Hypotheses

This thesis is concerned with representing and reasoning about tolerance statements in design. More generally, the thesis is about design. Design is an important kind of intelligent behaviour and is thus rightly the concern of Artificial Intelligence researchers. As AI scientists, the aim is to develop and deepen our understanding of the design process. As engineers, the aim is to use these new insights to implement tools which improve upon the geometry-based computer-aided design systems currently available. These twin aims, stated as hypotheses, motivate the work reported here.

1.2.1 Improving Support for Human Designers

Hypothesis 1 *Support for human designers can be improved by moving from a geometry-based approach towards an AI-based approach to engineering design.*

Commentary

The hypothesis aims to suggest that AI-based design *improves* support for human designers beyond that provided by geometry-based design. There is no attempt to argue that AI-based design represents the ultimate solution to the task of supporting human designers. In addition, AI-based design itself is not introduced as a vague term to be characterised *a posteriori* from the answers to the perceived deficiencies of geometry-based design. Rather AI-based design has the characteristic properties described in Chapter 2: automated reasoning, knowledge representation, intelligent control and consistency maintenance.

1.2.2 Delivering Improved Support to Human Designers

Hypothesis 2 *Design Support Systems constitute an effective means to deliver AI-based support to human designers.*

Commentary

This hypothesis is interesting only if there is evidence that the AI-based design approach improves support for human designers. In that sense, the hypothesis is clearly secondary to that of Section 1.2.1. However, by explicitly referring to Design Support Systems (DSSs) and their *effective* use, the hypothesis is clearly testable. The characteristics of DSSs are described in Chapter 2.

1.2.3 Experimental Objectives

From these hypotheses and the characterisations of AI-based design and DSSs in Chapter 2 a set of more specific and testable experimental objectives can be derived (see Chapter 3). These objectives thus inform the experimental work and serve as a basis for assessing results. Note however that the hypotheses place no requirements on the engineering design environment. In practice, before valid objectives can be derived the design and product creation context within which these objectives are situated must be described.

1.2.4 Design and Product Creation Context

The product creation environment used in this thesis is mechanical engineering. More specifically, the examples and observations are drawn from design and manufacturing in the field of diesel fuel injection equipment at Lucas Automotive Ltd.

Characteristic differences exists between design domains. For example, Dixon identifies four issues which distinguish mechanical engineering from electrical circuit design: range of material choice; sensitivity to manufacturing concerns; non-modularity of mechanical designs; and the role of complex 3-D geometry [Dixon, 1986]. Clearly, generalisations beyond the experimental domain about the improved support provided by the AI-based design approach and the effectiveness of Design Support Systems must be made with caution.

Important differences also exist in the kinds of environments in which products are created. In small craft industries individuals may be responsible for both design

and manufacture whilst large businesses may undertake design work, but contract manufacture to a different organisation in another country. The kind of product creation environment and its commercial and financial considerations are directly relevant to the engineering design process and the individual designer [Carter, 1990]. As with variations between design domains, generalisations must be made carefully. For example, the ability to share intellectual capital will differ within and between commercial organisations. In the case of Lucas Automotive, complex or strategic components are typically manufactured 'in-house' and standard parts bought from suppliers so the effect of commercial confidentiality will vary between components in a designed artefact.

1.3 Thesis Outline

The thesis is composed of seven chapters and two appendices. Chapters 1 and 2 describe the motivation and background to the thesis. Chapter 3 presents the experimental objectives of the thesis and the apparatus to be used in testing and investigating them. Chapters 4, 5 and 6 describe the experiments themselves with the results being summarised in Chapter 7.

The remainder of this outline is a chapter-by-chapter summary of the thesis contents:

Chapter 1 outlines the motivation for the thesis. AI-based design and tolerance representation are introduced.

Chapter 2 describes related work in the fields of AI-based design and tolerance representation. In particular, the elements of AI-based design and the attributes of Design Support Systems (DSSs) are presented. These characterisations underpin the thesis hypotheses.

Chapter 3 presents the experimental objectives derived from the motivating hypotheses of the thesis. The use and manipulation of tolerance statements in design is presented as a valid domain in which to pursue these objectives. The second half of Chapter 3 introduces the Edinburgh Designer System (EDS), an

experimental DSS. Several ways of using EDS for design problem-solving are illustrated and the existing facilities for tolerance representation in EDS are described and discussed.

Chapter 4 addresses the process by which the effect of a number of tolerance constraints taken together is inferred. Three tolerance combination techniques are described: functional loop analysis, inference from degree-of-freedom constraints, and analysis of tolerance distribution. The implementation of tolerance combination support within DSSs, specifically EDS, is examined.

Chapter 5 addresses the process of assigning tolerance statements during design. Two approaches to tolerance allocation are described: the use of standards and experience being contrasted with the systematic use of a quality loss function. The implementation of tolerance allocation support within DSSs, specifically EDS, is examined.

Chapter 6 examines the improved provision of support for human designers within the wider context of the product creation process. Several approaches to integrating the application of computer-based systems to product creation are described and the use made of design descriptions investigated. An example of how a design description which is not geometry-based can be used as the basis for deriving consequences of use outside the design environment is presented. Specifically, the generation of parts list information from the EDS Design Description Document is demonstrated.

Chapter 7, which represents the conclusion of the thesis, summarises the results of the investigations into the support of the tolerance combination and tolerance allocation activities and the situated design process. The chapter also contains a list of the original contributions of the thesis, implementation details relevant to the work (including its current status) and some ideas for further work in this research area.

Chapter 2

Related Work

This chapter describes how the problem of representing and reasoning about tolerance statements in computer-based systems arises in the product creation process, and characterises aspects of this problem. This is done through discussion of work in related fields and description of previous solutions or circumventions of the tolerance representation and reasoning problem.

The related work described here falls into two main categories: AI-based design and tolerance representation. In addition, the use made of tolerance statements by computer-aided process planning systems and similarities with the problem of coping with uncertainty in assembly planning are briefly discussed.

2.1 AI-based design

The interest of AI researchers in design, as opposed to abstract problem-solving, arguably began with the publication of Simon's "The Sciences of the Artificial" [Simon, 1969]. In the chapter entitled "The Science of Design" the main issues pertinent to the study of design, and by implication to the application of AI techniques in that domain, are discussed. Many of those topics are still active research sub-fields today. Examples include the representation of design problems, and the need to find satisfactory solutions when optimisation is not possible.

Amongst the first published attempts to apply AI techniques to computer-based systems for design were [Latombe, 1976] and [Sussman, 1977]. Latombe used a hierarchical representation for the designed artefact description, and Sussman incorporated analysis by propagation of constraints along with dependency-directed backtracking to control search within the space of possible designs: these methods are still in widespread use today.

As research into AI-based design has proceeded, several distinct, but related, themes have arisen: modelling the design process, the development and application of methods to support or automate design activities, and the development of Design Support System architectures. In this thesis attention will be concentrated on Design Support Systems (DSSs), and in particular the Edinburgh Designer System and its underlying model of design (see Chapter 3). A bibliography of AI-based design is provided by [Duffy, 1987]. Research in the domain of engineering design, the context for this thesis, is reviewed in [Finger & Dixon, 1989a; Finger & Dixon, 1989b]. Several common elements have emerged from the various 'AI in design' research areas and taken together these can be said to characterise AI-based design. The following sections outline these characteristic features drawing on a comparison of 'geometry-based' and AI-based design by Smithers [Smithers, 1989].

2.1.1 Elements of AI-based Design

- *Automated Reasoning.* CAD systems typically provide several automated reasoning functions. These are limited to forward inferencing — usually procedural — methods for performing geometric calculations or manipulating the appearance of the geometric model [Rooney & Steadman, 1987]. Previous attempts to fully automate the design function have concentrated exclusively on extending this forward inferencing capability. This approach amounts to an attempt to substitute a computer-based system for the human designer between the product concept and process planning stages of the CAD/CAM model.

A second type of automated reasoning is sometimes called backward chaining. Several production systems for evaluation are of this kind (see Finger and Dixon's review of work on *Analysis in Support of Design* in [Finger & Dixon, 1989b]). These programs are more widespread in domain specific applications, and play no role in the usual CAD/CAM model of computer-assisted product creation. The development of backward chaining for design evaluation was one of the first indications that the models underlying AI-based design were less constrained than those implicit in CAD.

The combination of both forward and backward chaining within a single architecture is a feature of design support systems (see Section 2.1.2 below). The lack of adequate knowledge representations to support activities within the design process limits the application of these automated reasoning strategies. For example, current systems cannot easily reason with incomplete geometric representations, though this is something that human designers appear to do when they sketch[1].

Another distinct type of automated reasoning is the use of grammatical formalisms to provide the transformational rules for building representations of designed artefacts. An introduction to this field is provided by [Mullins & Rinderle, 1991]. Machine learning is also being investigated as a source of automated reasoning for design problems. At Carnegie Mellon University, researchers are applying the SOAR system which "learns" by generating new chunks of knowledge from solved sub-problems to the synthesis of design solutions [Westerberg *et al*, 1989].

- *Knowledge Representation.* Adequate knowledge representations have long been seen as a major issue in AI-based design. Representations need to be

---

[1]Efforts to support reasoning about tolerance constraints during design have also had limited success. Finding adequate representations within a design description for the role tolerance statements have in manufacture, inspection, and assembly has proved problematical and is a major motivation of this thesis.

found for: the design problem itself (a requirements statement), the domain knowledge used in any subsequent design synthesis, and the proposed solution. The current industrial manifestations of the solutions to these problems are: sketches and customer documentation as a requirements statement; textbooks and human experience for domain knowledge, and an engineering drawing as the embodiment of the completed design description[2].

Many of the knowledge representation schemes developed by AI researchers have been applied to design problems. Production rules are commonly used in domain knowledge bases such as those for improving the quality of designed artefacts with regard to a particular domain at the detailed design stage; so-called 'design-for' activities (for example in [Swift *et al*, 1984]). Completed designs have variously been represented as: hierarchies of frames, sets of constraints, and graphs. Indeed, several systems (for example ALADIN [Rychener *et al*, 1986]) use multiple representations in order to better represent the different kinds of knowledge used in design[3].

One major study of knowledge representations for design is Gero's work at the University of Sydney on *prototypes* [Gero *et al*, 1988; Gero & Rosenman, 1990]. Prototypes are frame-like representations which can be used to support different kinds of design problem solving. Design processes are classified by how a prototype is used: prototypes can be refined (routine design), adapted (innovative design), or generated (creative design). The use of the Edinburgh Designer System's knowledge representation scheme is compared to Gero's use of prototypes in Chapter 3.

---

[2]The application of computer-based systems to the product creation process has progressed haphazardly against this background, with the engineering drawing (often abstracted to nominal geometric data) retained as the common currency and means of exchange between activities within this process.

[3]The fact that multiple representations have been found useful in domain dependent systems — in this case alloy design — suggests, correctly, that they will prove necessary for more domain independent systems such as those for design support.

Whilst AI-based design is not geometry-based its successful application will require adequate geometric representations. Current geometric representations need to be extended to include variational as well as nominal data [Juster, 1992]. In addition, representations should address those features inferred by human users from engineering drawings which, allied to domain knowledge, make drawings the current industrial design description.

- *Intelligent Control and Problem Solving.* For systems which attempt to automate design as a heuristic search through a space of possible designs, intelligent control centres on preventing *"unproductive design efforts"* possibly through monitoring the system's own performance (as in the Dominic II system [Orelup *et al*, 1988]). By contrast, for Design Support Systems, the problems of intelligent control include not only minimisation of computation, but the effective coupling of the computer and human designer to produce a problem solving partnership [Fischer & Nakakoji, 1991].

- *Consistency Maintenance.* Although consistency maintenance could rightly be categorised as one aspect of intelligent control, its importance in AI-based design, particularly Design Support Systems research, merits its separate consideration. Consistency maintenance is necessary if design descriptions are to be generated which are consistent with the requirements statement motivating the design process. Human designers additionally require consistency maintenance mechanisms to support their consideration of incompatible alternative proposals during the design exploration process. There are three approaches to handling consistency maintenance within AI-based design:

  1. *Implicit Maintenance.* For researchers who model design as search, the space of possible designs is a set of internally consistent though mutually conflicting constraint sets. As the space of possible designs is searched, only consistent sets of constraints are encountered. Use of this implicit consistency maintenance is thus limited to well-understood domains where the necessary search spaces can be constructed in this

form. LOOS, an automated system for generating consistent spatial arrangements, uses consistency maintenance of this type: only well-formed solutions are generated throughout the search process [Coyne, 1989].

2. *Single Context Maintenance.* Several design support systems use the human designer to resolve inconsistencies as they arise due to constraint addition, deletion or modification. Through this mechanism the current design description is guaranteed to be consistent, and — as consistency is explicitly represented — can also serve to focus constraint modification toward preserving desirable consistencies. Recent examples of this type of consistency maintenance are in the constraint programming language Galileo2 [Bowen & Bahler, 1992] and TEST, a system to support design for testability [Kim *et al*, 1992].

3. *Multiple Context Maintenance.* The use of Truth Maintenance Systems such as de Kleer's Assumption-based Truth Maintenance System (ATMS) [de Kleer, 1984] enables self-consistent yet mutually inconsistent sets of constraints to be represented, allowing the designer to assess and compare designs without the limitation of having to think only in terms of a single current design context. Forbus and de Kleer argue that the basic ATMS is inappropriate for design problems where satisfactory solutions must be found through the exploration of a sub-space of possible designs. They propose the use of a *focus environment* to control inference towards design goals [Forbus & deKleer, 1988]. An alternative proposal, the use of *assumption-based context management*, for controlling inference within a design support system is presented in [Logan *et al*, 1991]. The Edinburgh Designer System uses an ATMS to maintain its Design Description Document (see Chapter 3).

The practical importance of consistency maintenance is generally accepted. Indeed, design offices consume time and effort ensuring, for example, that drawing issue numbers, parts list numbers and product batch numbers are consistent. Computer systems have been successfully employed to handle

this task more effectively, but the use of computer-based consistency maintenance during individual component or sub-assembly design has not been developed. Consistency maintenance can be expected to be an increasingly active element of both AI-based design and future CAD developments.

2.1.2 Design Support Systems

The development of system architectures for computer-based design support is a comparatively new research field. Its origins can be usefully traced back to the initial experiments with CAD systems and interactive computer graphics in the early 1960s. Since then, advances in CAD have been primarily in the construction of successively more 'realistic' visualisations and in the integration of numerically-controlled machining through common geometric data (typically a cutter location file). The provision of tools in addition to those provided by the CAD terminal at 'design time' has been largely, and regrettably, neglected. AI techniques can provide some of these missing tools: Swift's 'design for' advisers are an example [Swift *et al*, 1984]. Design Support Systems (DSSs) represent a more ambitious attempt to provide collections of useful tools and sub-systems suitably organised for the human designer[4].

DSSs can usefully be characterised by the following attributes:

- *Adoption of an AI-based approach to design.* DSSs are one of the kinds of computer-based system developed using the AI-based design approach. That is to say DSSs inherit the characteristic elements of AI-based design outlined in the previous sub-section: automated reasoning, knowledge representation, intelligent control and consistency maintenance.

- *Support for human designers.* Consideration of the complexity of problems facing the development of tools to *do* computer-based engineering design

[4]Suitably organised meaning with primary regard to the task undertaken rather than the tools available.

has led to two distinct strands of AI-based design research. The first approach tries to build systems which are able to create design descriptions from a given specification. Typically these systems are application-oriented, but they necessarily operate in domains where the space of possible designs is well-understood and can serve as a search space for the design problem solver. The second approach differs markedly from the first, by accepting the problems of using current AI techniques to undertake engineering design tasks and placing the emphasis on supporting the human designer: "Working with *the designers rather than* working for *them*" [Smithers, 1989]. DSSs are those AI-based design systems which adopt this second approach. Thus, unlike automated design systems, DSSs do not actually design anything rather they provide tactical and strategic support to a human designer exploring a space of possible designs [Logan *et al*, 1991].

- *Use of design process models.* As DSSs necessarily involve the human designer, it is not possible to avoid the problems associated with a lack of understanding of how humans design. For this reason, DSS builders are developing models of the design process in parallel with their system implementations. Experiments thus serve to test aspects of these models whilst the models inform and underpin elements of the support system implementations. A relationship between implementation and design process model of this type holds for the Edinburgh Designer System and its exploration-based model of design [Smithers *et al*, 1989]. A similar relationship holds between top-down refinement plus constraint-based reasoning model of design and the VEXED system at Rutgers University [Tong, 1990]. In the latter case the researchers intend to extend their model of design to underpin both automated design systems, such as VEXED, and design support systems.

- *Independence from specific design domains.* Arising from the interest in models of design has been a desire to develop a general understanding of design. This has led to the specification of system architectures which are not application dependent. Although within the field of research concerned with automating design there have been recent attempts, notably work on the

Dominic systems [Howe *et al*, 1986; Orelup *et al*, 1988], to introduce domain independence, acceptable domains are still constrained by the requirement to construct search spaces. DSSs hold the promise of greater domain independence through a coupling of human and artificial intelligence and a consequent ability to explore design possibilities without the prerequisite of constructing a searchable space of possible designs.

2.1.3 Discussion

In this section the elements of AI-based design and in particular of Design Support Systems (DSSs) have been described. The aim has been to distinguish AI-based design from the geometry-based approach adopted by the current Computer-Aided Design (CAD) industry. In addition within AI-based design, DSSs have been distinguished from attempts to automate design.

One might ask, is there any direct relationship between geometry-based CAD systems and DSSs? Perhaps surprisingly, the answer is yes. Both CAD systems and DSSs are necessarily interactive and rely on the effective presentation of the results of any automated reasoning. In contrast, design automation research concentrates on improved methods of automated reasoning such as the use of 'modification operators' to expand sets of prototype solutions [Murthy & Addanki, 1988]. Within the requirements for presentation and interaction, CAD research aims, for example, at improving the *coverage* of geometric shapes which can be easily manipulated (see Figure 2–1). DSS research aims to understand how to effectively support human designers. Therefore, both automated reasoning and improvements in visualisation are important. However, effective coverage within DSSs needs to extend beyond the geometry of the design description to include alternative representations useful to the human designer. Structures relating the functions of features within the design description might be one example [Pahl & Beitz, 1988]. In practice, although the relative emphasis of CAD and Design Automation research is clear from Figure 2–1, improving geometric coverage and

visualisation requires improved reasoning capability and the axes shown are not strictly orthogonal.

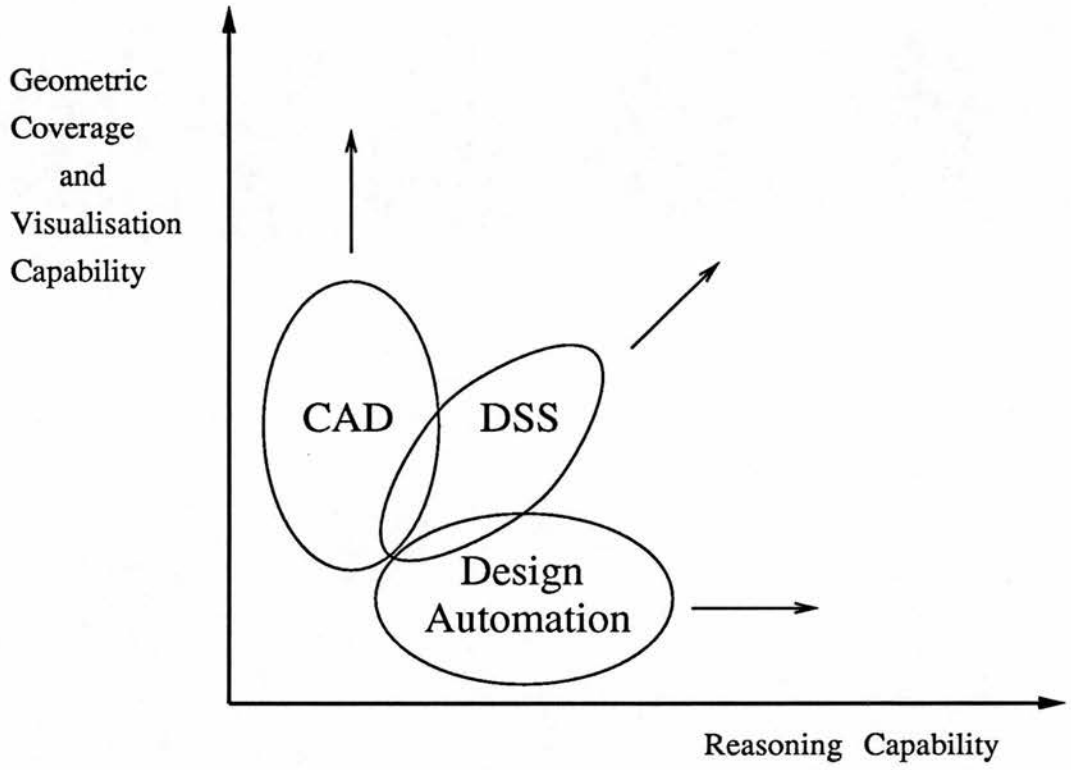


Figure 2–1: Research Directions in CAD and AI-Based Design

Missing from this account of design support is the relationship between those activities considered part of the design process and other activities within the process of product creation. Exceptions to this criticism are attempts to use manufacturing knowledge within detailed design and systems which aim to produce manufacturable design descriptions rather than nominally satisfying an initial functional requirement alone. An important example of this failure to consider product creation activities outside design is the representation of design parameters as nominal values without tolerance values. The manufacture of artefacts to satisfy a nominal design description without tolerances is impossible.

2.2 Tolerance Representation

Tolerance statements describe the allowable variations in location and form of instances of designed artefacts. Tolerance statements have a variety of uses in the process of product creation: for example influencing the cost of machining a component or the failure rate in assembly. Research has concentrated either on expanding computer-based representations from nominal dimensions to include tolerances and other variational data (see Section 2.2.1 below) or on problems associated with reasoning about tolerance representations; for example: finding the effects of combinations of tolerances and their distributions (the subject of Chapter 4). Most of the work on representing and reasoning about tolerance statements pays scant attention to the process within which their contribution is presumably situated. The probable cause of this deficiency is the preponderance of the 'compartmentalised' CAD/CAM model introduced in Chapter 1 which treats tolerance statements as an adornment to the nominal geometry model required[5] prior to automated process planning. One notable exception to this criticism is Rehg *et al*'s tolerance allocation work within automated design synthesis [Rehg *et al*, 1988]. The work reported here shares their view of tolerance statement reasoning as situated within product creation, but adopts a Design Support System approach rather than a design automation approach.

Later chapters in this thesis examine the major reasoning tasks undertaken with tolerance representations: combination and allocation. The relationship between this work — within a Design Support System architecture — and geometric modellers and the models they produce is an uneasy one and likely to remain so. As Arbab notes, problems are unavoidable in shape-oriented models *"where the fundamental protagonists (ie. manufacturing operations) are not recognized, represented and treated properly"* [Arbab, 1982]. Of course manufacturing operations

[5]No doubt to the annoyance of the CAD/CAM model developers!

in fact form only one class of protagonists, others arise from assembly, inspection, and maintenance requirements. Notwithstanding these criticisms, previous work on expanding geometric representations has proved useful both theoretically and practically in providing more formal representations for tolerance statement reasoning tasks. This work is the subject of the following sub-section.

2.2.1 From Nominal to Variational Geometry

Historically, the primary concern of computer-aided design (CAD) has been with the production of engineering drawings. Typically, this process begins with the construction of a part database to represent the nominal geometry of the designed artefact. Additional information such as tolerances, surface finish data, and the design notes are attached to the nominal geometry, but this attachment process is often quite crude. For example, a tolerance may be considered a *"draw mode"* entity within the two dimensions of a drawing sheet whilst the geometry to which it refers is a *"model mode"* entity in three dimensions. When drawing production is considered the ultimate aim of the design process such limitations are inconsequential: difficulties arise when CAD data is required for activities such as computer-aided process planning.

Work on extending geometric models to include the meaning of current tolerancing standards aims to define variational models which represent the set of designed artefacts instances which satisfy a known tolerance constraint. (An excellent review of work on tolerance representation is provided by [Juster, 1992].) Three main approaches to developing variational models have been attempted: object parameterisation, variable vector spaces, and offset modelling [Juster, 1992].

Object parameterisation aims to define the variational class modelled as a function of the model parameters. When the designed artefact dimensions are defined in terms of model parameters this parameterisation is said to be direct. If dimensions are derived from parameters, such that changes to dimensions require parameters to be re-calculated, then the parameterisation is said to be indirect. Direct parameterisation is easier to implement, but constrains the designer to

model the designed artefact in terms of its toleranced dimensions only. In addition, some geometric tolerances cannot be modelled. For example, the perfect form primitives used within variational models based on constructive solid geometry cannot support tolerance data. Thus, cylindrical primitives cannot represent their own cylindricity.

Indirect parametrisation requires the inversion of the known relationship between dimensions and parameters to define the variational model in terms of model parameters. The advantage of this approach is that the designer's assumptions are the starting point for the definition of the variational model. The disadvantage is that the inversion of the relationship between dimensions and parameters is non-trivial. The earliest work on extending geometric models to represent tolerance information, Hillyard and Braid's paper on *"characterizing non-ideal shapes"* [Hillyard & Braid, 1978], adopts the indirect parameterisation approach.

For Hillyard and Braid an object is visualised as a pin-jointed infinitely elastic wire frame covered by elastic membranes, whose dimensions are subsequently constrained by the human designer. Variations in the positions of the wire frame vertices are related to the rigidity matrix for the wire frame and the variational class is found by inverting the rigidity matrix to form a flexibility matrix. The work of Hillyard and Braid has been generalised by Light to support the modification of underlying geometry through changing dimension values and the efficient inversion of the dimension-parameter relationship [Light & Gossard, 1982].

One alternative to object parameterisation is the construction of vector spaces to directly represent allowable variations in model variables. This method, proposed by Hoffman, leads to tolerances being calculated from a set of inequalities, known as tolerance functions, on model variables and the variational class being defined by a region of the model vector space [Hoffman, 1982]. Unfortunately, finding the region within the vector space which defines the variational model is problematical: tolerance functions which support current tolerancing practice and generate provably bounded vector spaces still remain to be defined. Another problem with the vector space approach is that tolerances of form, such as flatness, cannot be represented within a finite vector space. The adequate representation

of tolerances of form is one of the advantages claimed for offset modelling and the theory of geometric tolerance proposed in [Requicha, 1983a].

Requicha suggests that rather than attempt to assign attributes to the imperfectly-formed objects which arise from real world operations bounds should instead be placed on perfect-form objects. In practice, geometric tolerances can then be represented as zones located and oriented around perfect-form objects. One consequence of this approach is that tolerance types can be characterised by how these tolerance zones are constructed relative to the perfect-form object. In other words, Requicha suggests the use of a small set of more general tolerance types such as form, size, and orientation rather than the more numerous and specific classifications such as roundness and concentricity defined by the engineering standards organisations (for example, the British Standards Institution [BSI, 1984]).

Figure 2–2 shows how an arbitrarily positioned tolerance zone between an upper bound (U) and lower bound (L) of thickness Δf represents a form tolerance. This is equivalent to a flatness or cylindricity tolerance for the cases where the perfect-form objects are a plane or cylindrical surface respectively.

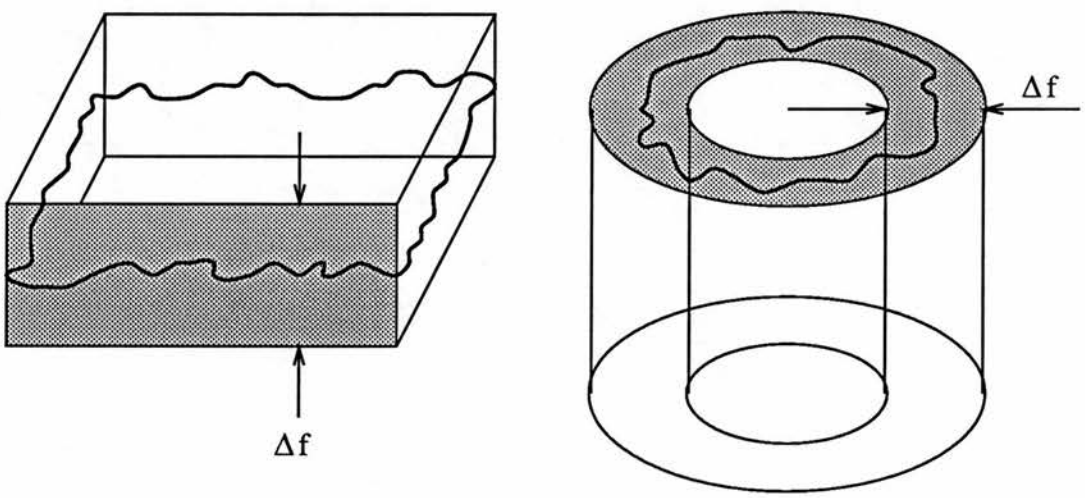


Figure 2–2: Flatness and Cylindricity as Form Tolerances

An additional consequence of the more formal nature of Requicha's theory is

that several widespread tolerancing practices — conventional +/− tolerancing and the use of implicit datums for example — are shown to produce ambiguous or incomplete representations. Indeed, a major attraction of Requicha's approach is to manage the complexity associated with the flexibility of current tolerancing practice. As an example of how complicated current practice is, the tolerance input routine of a recent process planning system *"displays twenty-one items of toleranc-ing information about each surface at a time"* [Wang, 1987]. There are a number of unresolved issues associated with offset modelling. Firstly, as Requicha's theory departs from current practice, evidence is required that the functional requirements of a variational model can all be supported. In addition, the specification of valid tolerance zones by human designers has yet to be demonstrated. (Requicha and Chan propose one method of implementing Requicha's theory within a constructive solid geometry modeller [Requicha & Chan, 1985].)

Further evidence for the utility of Requicha's approach is provided by the work of Fleming in connection with the analysis of uncertainty in an assembly of parts [Fleming, 1987]. Fleming adopts the geometric tolerance formalism proposed by Requicha without the latter's emphasis on solid modelling, [Requicha, 1983b], to characterise an assembly as a network of tolerance zones for features of constituent parts. Fleming's tolerance combination method, its application to design, and its possible implementation within a design support system framework are investigated in Chapter 4.

2.3 Other Work

In this section the uses to which tolerance representations are put within the product creation process are discussed. The main technical determinants of product creation effectiveness are: product design; process design; facilities design and activity planning [Voelcker *et al*, 1988]. Reasoning about tolerance statements is an important part of both product and process design, but may also influence facilities design and activity planning.

Supporting tolerance statement reasoning in design, for example techniques for tolerance combination, is a major subject of this thesis (see Chapters 3, 4 and 5). Process design for a designed artefact is also fundamentally affected by tolerance statements as these constraints define the allowable dimension variation in production. Within process design three main process classes can be identified: part-manufacturing processes, assembly processes and inspection processes [Voelcker *et al*, 1988]. Tolerance statements within design descriptions influence all these activities.

2.3.1 From Variational Geometry to Process Plan

Process planning is the act of preparing a detailed plan for the production of instances of a designed artefact. The major approach adopted by researchers within the field of computer-integrated manufacture has been to attempt to develop computer-based systems to automate the creation of the process plan from a geometry-based design description: so-called generative process planners.

Generative process planning involves the synthesis of the plan from a set of input data including the designed artefact and raw material description, and available manufacturing processes and tools. Process planning typically proceeds through feature recognition, set-up planning, process selection and process ordering activities [Voelcker *et al*, 1988]. Most process planners reported in the literature do not address the entire process planning problem, for example, input design descriptions are often assumed to be already decomposed into manufacturing features [Requicha & Vandenbrande, 1988].

Tolerance representation within process planning is in two major forms: the input design description contains tolerance constraints on features of the design; and the available manufacturing processes outline the suitability of machining operations for achieving the geometric attributes required for that feature. The successful 'matching' of process capability to tolerance requirement, the process selection activity, is one of the key steps in determining the process plan. In

practice, tolerance representations are also required to support reasoning in set-up planning and process ordering.

Many process planning systems have been developed and an extensive discussion of them can be found in [Chang & Wysk, 1985]. An introduction to process planning and its relation to problem-solving in AI can be found in [Joshi *et al*, 1986] and a review of the integration of geometry-based design and process planning is provided by [Chang, 1990]. The first application of AI to this domain was GARI, a part machining planner, described in [Descotte & Latombe, 1981]. GARI's tolerance representation is within the *entities* which characterise features of the design to be machined and within the statements which define the available machines. For example, Figure 2–3 shows for one feature, a countersunk hole, how conventional +/− tolerancing on dimensions and between faces is supported. In addition, rules which govern the formation and ordering of the plan use geometric tolerance relationships between entities to constrain the matching process between entity and machine. For example, *"if &x and &y* (two entities) *are two bores linked by a coaxiality constraint, then one is advised to execute their finishing cuts in the same phase on a lathe"* [Descotte & Latombe, 1981].

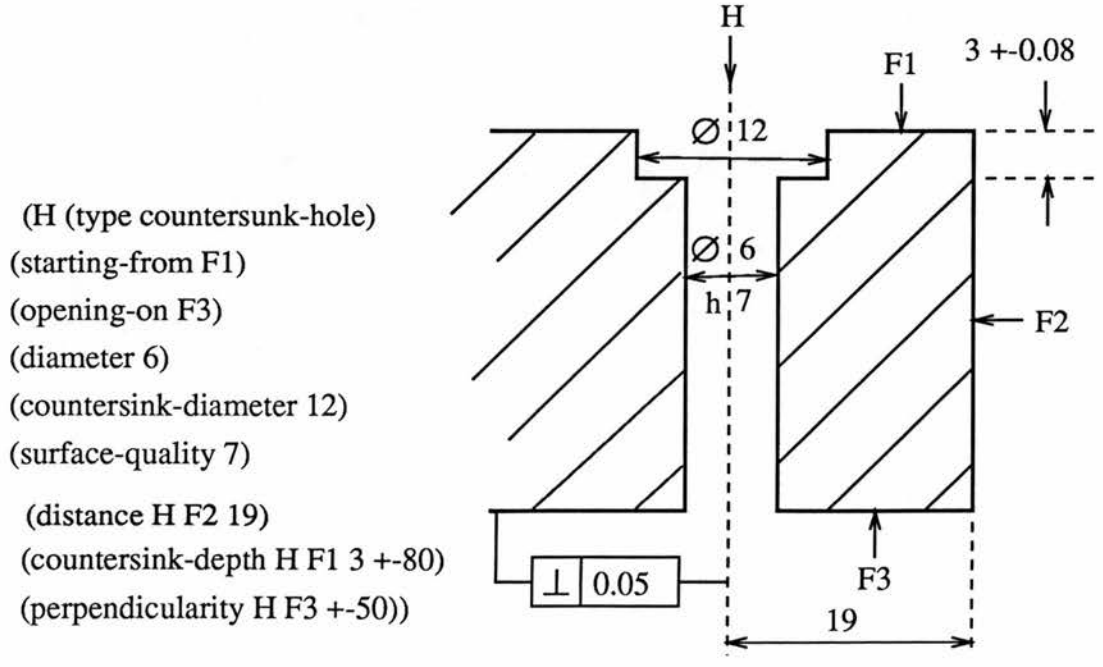


Figure 2–3: Tolerance Representation within GARI's Entities

Several other Computer-Aided Process Planning systems have adopted and developed GARI's approach to tolerance representation. For example, Nau uses frames to characterise machining processes so that tolerance constraints are a *"restriction"* slot to be satisfied [Nau, 1987]. Where geometry-based, rather than manufacturing feature-based, input design descriptions have been used tolerance representation is often ignored and process planning becomes the generation of cutter paths from nominal geometry (Mayer *et al*'s IMPA is a recent example [Mayer *et al*, 1992]). More realistically, when only nominal design descriptions are available, tolerance data must be appended manually. Wang's *"tolerance input routine"* is an example of this approach [Wang, 1987].

2.3.2 The Ubiquitous Tolerance Statement

Tolerance statements are salient constraints for activities apart from design and process planning. For example, the tolerance vocabulary used within a design description is also required in the specification of facilities for production and inspection. Indeed, tolerance constraints are the primary dictator of inspection or gauging process requirements[6]. Similarly, decisions affecting the maintenance of instances of the designed artefact in service are dependent on tolerance data. For example, considerations as to which parts or sub-assemblies are to be replaceable during the product's life are typically tolerance statement-based.

A further area of interest is that of uncertainty in assembled parts, particularly with regard to robotic assembly planning. Fleming's thesis [Fleming, 1987] tackles this problem providing answers to questions such as *"can these parts go together?"* and *"what is the maximum 'slop' when these parts have been assembled?"*. Durrant-Whyte's paper on uncertain geometry [Durrant-Whyte, 1986] considers the distribution of instance dimensions within a tolerance range and uses tolerance combination inferences to find consequent dimensional distributions with the aim

---

[6]In Lucas Automotive, specifications for measuring equipment are typically based on the requirement to be accurate to 10% of the tolerance allowance.

of using these results to influence robotic task planning and execution. It is the author's belief that analysis of this type should be carried out within the design process where results can effectively influence, for example, the datum system specification. Consideration of uncertainty in assembly is properly a concern of the design process.

Finally, several fields within AI make use of techniques similar to those used in the analysis of tolerance statements within AI-based design. In addition to robot planning (mentioned above, but including work such as [Brooks, 1984]), the analysis of results arising from approximate methods (for example in design synthesis [MacCallum *et al*, 1985]) will require techniques for representing and finding solutions within sets of simultaneous inequalities such as those which arise during tolerance analysis.

2.4 Summary

This chapter has presented work related to the subject of this thesis with a view to demonstrating the source and nature of the tolerance representation problem as it arises in the developing field of AI-based design. In particular, AI-based design and Design Support Systems (DSSs) were characterised in terms of their major elements or attributes (Sections 2.1.1 and 2.1.2 respectively). Work on tolerance representation and reasoning was reviewed in Sections 2.2 and 2.3. This work was shown to be typically based on the sequential model of product creation: the nominal geometric design description serving as an *idée fixe* for subsequent activities such as process planning. The relationship between the many uses of tolerance statements and DSS research within AI-based design is the subject of the next chapter.

Chapter 3

Problem Scenario

This chapter brings together the work on AI-based design and tolerance representation described in the previous chapter. The chapter has three sections.

In the first section experimental objectives derived from the motivating hypotheses of the thesis are presented. The objectives are: to show how AI-based design descriptions can be used to support reasoning about tolerance statements; to show the application of an AI-based design description within the product creation environment; and to demonstrate the delivery of this support through an experimental Design Support System.

Reasoning about tolerance statements is the subject of the second section. The significance and validity of tolerance reasoning as a context for making comparisons between the AI-based and geometry-based approaches to supporting design is discussed. A list of tolerance reasoning tasks is presented. Tolerance combination and tolerance allocation are shown to be the major tasks which AI-based design must be shown to support.

The Edinburgh Designer System (EDS), an experimental DSS, is introduced in the third section and several ways of using it for design problem-solving are illustrated. In addition, the existing facilities for tolerance representation in EDS are described and discussed. Finally, the chapter closes with a brief summary of the ground covered.

3.1 Experimental Objectives

In Chapter 1 the twin hypotheses motivating this work were presented. The first hypothesis states that support for human designers can be improved by moving from a geometry-based approach towards an AI-based approach to engineering design. The second states that Design Support Systems constitute an effective means to deliver AI-based support to human designers.

Testing the first hypothesis requires the characterisation of geometry-based and AI-based design, and the identification of a valid and significant design activity for which the AI-based approach can be assessed. For this thesis reasoning about tolerance statements in design is the specific activity to be investigated. The justification for this choice is explored in Section 3.2.1.

An additional objective in testing the first hypothesis, and a pre-requisite for the adoption of an AI-based approach to engineering design, is to show that AI-based design descriptions can be used to support activities within the product creation process which situates design. Recall that geometry-based design descriptions form the basis for the CAD/CAM data flow model introduced in Chapter 1. The use of AI-based design descriptions is investigated in Chapter 6 with the specific objective of demonstrating the use of an AI-based design description to generate a parts list for use within the wider product creation process.

Testing the second hypothesis requires the availability of a Design Support System (DSS) and its use to demonstrate the delivery AI-based support. Specifically, the objective is to implement within an experimental DSS, the Edinburgh Designer System (EDS), the support for the tolerance reasoning activity and parts list generation required by the other experimental objectives.

Thus, the experimental objectives for this work can be summarised as:

- Show how AI-based design can be used to support reasoning about tolerance statements.

- Show how an AI-based design description can be used within the wider product creation process.

- Demonstrate the delivery of this AI-based support through an experimental Design Support System.

These objectives form the basis for assessing the results of the experiments in the concluding chapter of the thesis.

3.2 Reasoning about Tolerance Statements

3.2.1 Justification

What characteristic of the tolerance representation and reasoning task makes an improvement in the way that it is supported *significant*? In addition, why is tolerance representation a *valid* area to assess the geometry-based and AI-based approaches to design support?

Product creation involves the realisation of a design description in a designed artefact. Tolerance statements are significant because they relate the designed artefact to the design description. In other words, tolerance statements define within the design description the allowable variations (in location and form) of designed artefacts and serve as a basis for testing designed artefact instances against their design description. Thus, improving the way that tolerance representation and reasoning tasks are supported will directly benefit the process of realising design descriptions as designed artefacts.

The validity of tolerance representation as a test to assess the geometry-based and AI-based approaches arises from the role of tolerance statements within the design description. The development of 'informationally complete' geometric design descriptions is the principal goal of geometry-based design [Juster, 1992]. Previous work described in Chapter 2 explored how nominal geometry representations have been expanded to include variational data such as tolerance statements.

In addition, work on computer-aided process planning makes clear the importance of this variational data in the derivation of a plan for manufacture from this extended geometric design description. The design description and the tolerance statements it contains are also important within AI-based design. For example, in the exploration-based model of design to be outlined in Section 3.3.1, the design specification together with a final requirements description represent the direct results of the engineering design process. Thus, reasoning about tolerance statements is a significant area in which to examine design descriptions which are not geometry-based, and also allow a valid assessment of the two approaches to be made.

3.2.2 Tolerance Reasoning Tasks

It is tempting to develop experiments and tests for tolerance statements in AI-based design based on existing work on tolerance representation (some of which was reviewed in Chapter 2). A better strategy, and the one adopted here, is to examine the kinds of tolerance reasoning *task* undertaken by engineers within the product creation process. The following list was developed following three-month placements in the design and manufacturing departments of a division of Lucas Automotive Ltd. :

- Designers allocate tolerance constraints, including the datum systems on which the variational geometry is based, in the knowledge that this will influence and sometimes dictate the type of manufacturing method adopted.

- Manufacturing engineers analyse how tolerances on individual components combine to determine strategies for assembly.

- Tradition and experience are used to allocate tolerances. In addition, tolerances are often not re-evaluated when a design is revised: tolerance values are 'carried forward' into the new versions. (This observation — the informal way in which the tolerance specification task is undertaken — provided

the motivation for Rehg *et al*'s work on tolerances in AI-based design [Rehg *et al*, 1988].)

- Engineers make modifications to designs. Unless the relationships between the functional characteristics of the product and its variational geometry are recorded explicitly, minor changes in geometry — for example, to simplify manufacture — cannot easily be implemented. Making a modification therefore involves the re-derivation of this relationship between variational geometry and functional characteristics before the effect of any proposed change can be investigated.

- The ease with which a tolerance constraint is satisfied by a process is studied by manufacturing engineers. The distribution of geometry about the nominal value can be introduced as a measure of quality.

- Manufacturing engineers devise tools and methods for verifying that tolerance constraints allocated by the designer are satisfied in production.

- The design of tools and fixtures for holding parts during manufacture requires analysis combining the intrinsic uncertainty of the geometry of the fixture with the extrinsic uncertainty of the geometry of the part being held.

Two activities stand out as common across a number of these tasks: tolerance combination and tolerance allocation. The next section exposes how, in even the simplest example, a functional constraint and the requirement for tolerances on geometry gives rise to these twin activities.

3.2.3 Combination and Allocation Exposed

Figure 3–1 shows the base of a spark plug from a petrol engine. Following the method presented in the British Standards Institution guide to tolerancing functional dimensions [BSI Education, 1985], a *functional loop* is identified. In this case, the loop 'contains' a functional requirement on the width of the air gap, f.

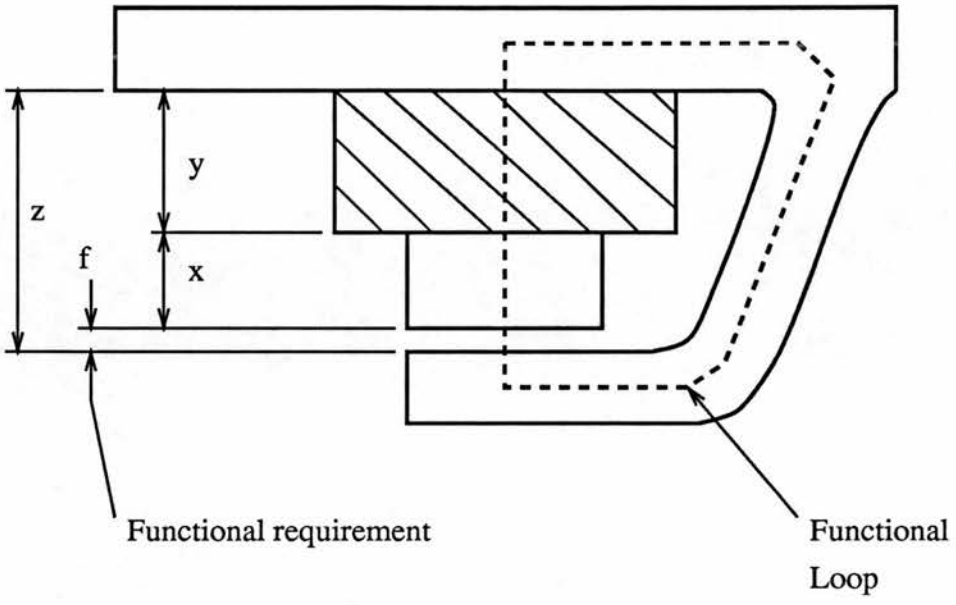


Figure 3–1: Example of a Functional Loop

Clearly, the constraint on the nominal values is:

$$f = z - y - x \tag{3.1}$$

To derive an equation relating the tolerance bounds on x, y, z and f, consider the limits on f:

$$\min f = \min z - \max y - \max x \tag{3.2}$$

$$\max f = \max z - \min y - \min x \tag{3.3}$$

Subtracting Equation 3.2 from Equation 3.3 gives:

$$\max f - \min f = (\max z - \min z) + (\max y - \min y) + (\max x - \min x) \tag{3.4}$$

Now the tolerance on any dimension D is given by:

$$\Delta D = \max D - \min D$$

Note that, as expected, $\Delta D \geq 0$. Substituting this definition of tolerance value into Equation 3.4 yields:

$$\Delta f = \Delta z + \Delta y + \Delta x \tag{3.5}$$

This is the variational equivalent of the nominal geometry constraint given as Equation 3.1. Note that the tolerance on the functional requirement, f, is equal

to *the sum of* the tolerances on the nominal values affecting the function. Clearly, more complex functional relationships, which need not be geometric, between parameters contributing to a functional requirement result in more complex variational constraints and Equation 3.5 does not apply.

Tolerance Allocation

Functional requirements, though incomplete or inconsistent, often serve as the starting point for the engineering design process (recall Figure 3–2). Thus, in general, the functional requirement, f, and its bounds Δf are known. For example, the air gap for the spark plug must be wide enough to produce consistent ignition timing and narrow enough to provide a strong spark. The tolerance allocation activity reduces to finding values for Δx, Δy and Δz which satisfy Equation 3.5 for the known Δf.

Tolerance Combination

Once the designer has allocated values to Δx and Δy tolerance combination can be applied to infer the value of Δz.

Re-arranging Equation 3.5 gives:

$$\Delta z = \Delta f - \Delta y - \Delta x \tag{3.6}$$

Note that $\Delta z \geq 0$. If this is not true then the complete functional tolerance Δf has been allocated to Δx and Δy. For the spark plug example, Δy and Δx are controlled in production and f is adjusted within the bounds of Δf. Thus, control of z within Δz is a *consequence* of this combination of tolerances.

3.2.4 Beyond Combination and Allocation

Tolerance combination and allocation are viewed as the two primary tolerance reasoning activities. Thus, the focus of the experiments is how the characteristics of AI-based design, such as automated reasoning, can provide improved support

to human designers performing these two tasks. However, a number of additional topics common to both combination and allocation are addressed:

- *Tolerance Distribution.* Tolerance statements specify the limits on geometric variation. Within these limits, instances of geometry have differing values. The likelihood, or probability, of an instance taking a particular value within the tolerance limits is represented as the tolerance distribution. For example, this probability might be represented as a normal distribution where the mean is equal to the nominal geometric value. To answer questions about the failure rate in assembly requires an understanding of the way that the tolerance distributions of the component parts are combined and allocated.

- *Quality Engineering.* How are the choices for the particular values for tolerance constraints arrived at? In the broad sense, the designer places bounds on the geometry to guarantee that the ultimate product is of an acceptable quality. Taguchi's work on *tolerance design* uses the concept of a Quality Loss Function [Taguchi, 1986; Taguchi *et al*, 1989]. To determine acceptable bounds on nominal values, the relationship between the cost of satisfying these tolerance constraints and the quality loss associated with any deviation from the nominal geometry has to be understood.

- *Analysis of Complexity.* The experiments involve simple examples. The practical application of the results, for example through the effective use of Design Support Systems, requires that the results remain valid for more complex cases. In other words, there has to be some analysis of computability for the reasoning methods and an investigation of the complexity of the problem domain. Thus, the ease with which simplified methods can be generalised must be examined. For example, what increase in complexity is associated with moving from one-dimensional to two-dimensional analysis? How much more for three-dimensions?

3.3 The Edinburgh Designer System

Research into AI-based design has been underway at the Department of Artificial Intelligence, University of Edinburgh since the mid-1980s. Originally, this work, and the work on the Edinburgh Designer System (EDS) in particular, formed part of the Alvey Large Scale Demonstrator Project *Design-to-Product* [Smithers, 1985]. Research has focussed on developing and testing experimental systems (successive versions of EDS) to support human designers in terms of the *knowledge process* that underlies the design task. Indeed, it is by building such systems that an understanding of design as an exploration process is being developed.

3.3.1 An Exploration-based Model of Design

The 'traditional' product cycle model described in Chapter 1 presents design within a set of stages and feedback paths in which data is said to flow. This model, currently the basis for the application of computer-based systems to product creation, leads to an emphasis on the need to communicate data rather than understand and integrate the knowledge used throughout. In short, the model does not help us to understand the knowledge process underlying activities within product creation. To develop this understanding, the exploration-based model of design shown in Figure 3–2 is being tested. (A full account of this model is given in [Smithers *et al*, 1989]. The following is only a brief description.)

In the model shown in Figure 3–2 the design process starts with an initial requirement description R_i. Typically, R_i is incomplete and may be inconsistent and so cannot be said to define a *goal-state* in a space of possible designs. Thus, the design process, E_d, from this starting point is not a search problem. Rather there is an exploration of the design space during which inconsistencies and incompletenesses in R_i are revealed. The requirement description is modified as the exploration continues until a point in the design space is arrived at that fully

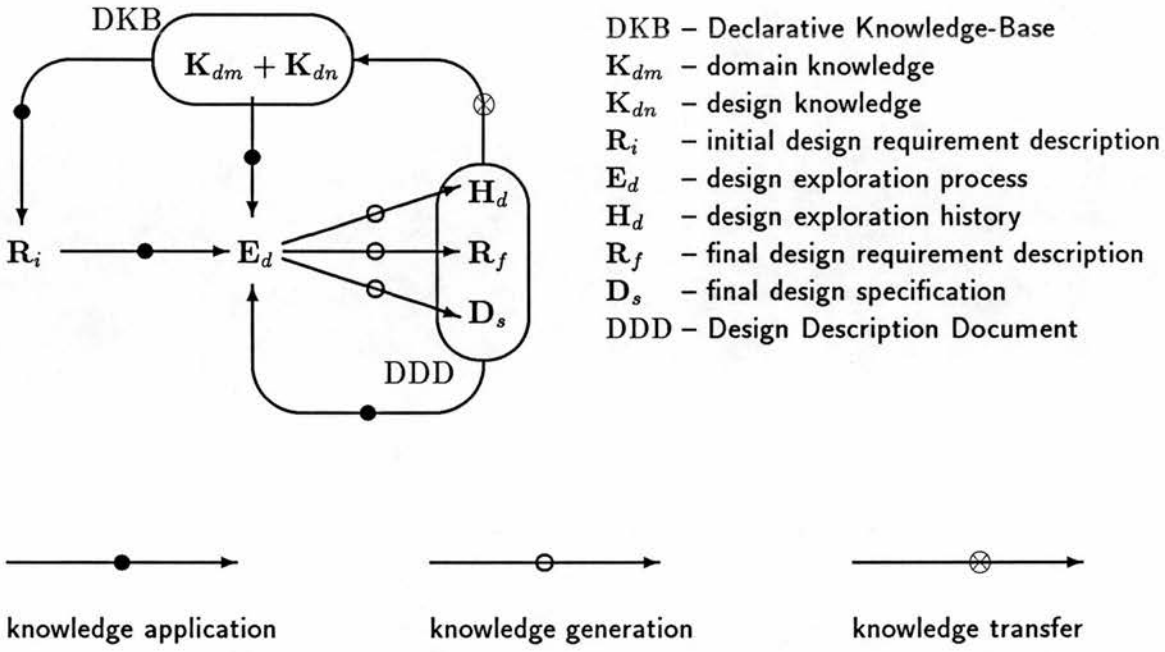


Figure 3–2: Exploration-based model of design

specifies a design which meets the evolved requirement description[1]. The result is a final consistent requirement description, R_f, and a design specification, D_s, which is consistent with it.

An important product of the design process is the design exploration history, H_d. This represents the history of the design expedition: what paths were taken in the design space and why. H_d is of course of vital practical consequence when reviewing, re-using or modifying the results of previous design tasks. Collectively, the final requirement description, the final design specification and the design exploration history constitute the Design Description Document.

The Declarative Knowledge Base, DKB, required to support the design exploration process is composed of Domain knowledge, K_{dm}, and design knowledge, K_{dn}. Domain knowledge is used to delimit the space of possible designs whilst design knowledge concerns how this space of designs can be explored. Typically, K_{dm}

[1]This is not an optimisation process. Instead we terminate our exploration at a satisfactory point. A process Simon called *satisficing* [Simon, 1969].

and K_{dn} are related. For example, many problem solving techniques cannot be sensibly expressed without reference to the domain in which they are applicable.

3.3.2 Architectural Overview

Work on EDS has mainly involved the implementation of an architecture in which a number of sub-systems can be integrated and controlled. The results served as experimental apparatus to improve our understanding of the design process and to test aspects of the exploration-based model. For the work presented here, EDS provides the experimental apparatus and the implementational background with which to test the hypotheses presented in Section 3.2.

The EDS architecture is shown in Figure 3–3. Central to this architecture is maintaining and controlling the development of the Design Description Document (DDD). The DDD, as its name suggests, contains the description of the design for the designed artefact (corresponding to D_s in our exploration-based model). It also constitutes the record of the design process (H_d in our model). This is possible because the DDD consists of a relational data structure based on the dependencies between items of information within it and maintained by an Assumption-based Truth Maintenance System (ATMS) [de Kleer, 1984].

Users develop their designs by making statements, called *assumptions*, via an interface. The system uses a collection of general inferencing sub-systems which act as knowledge sources and derive necessary consequences based on the designer's assumptions. Control of the interaction between these inferencing sub-systems is in the style of a Blackboard system [Hayes-Roth, 1985]. The DDD provides the common data structure through which the designer and support systems co-operate.

Automated Reasoning

Automated reasoning within EDS comes in two styles: general support sub-systems called Knowledge Sources and specialist support sub-systems called Design Support Toolkits (as shown in Figure 3–3).

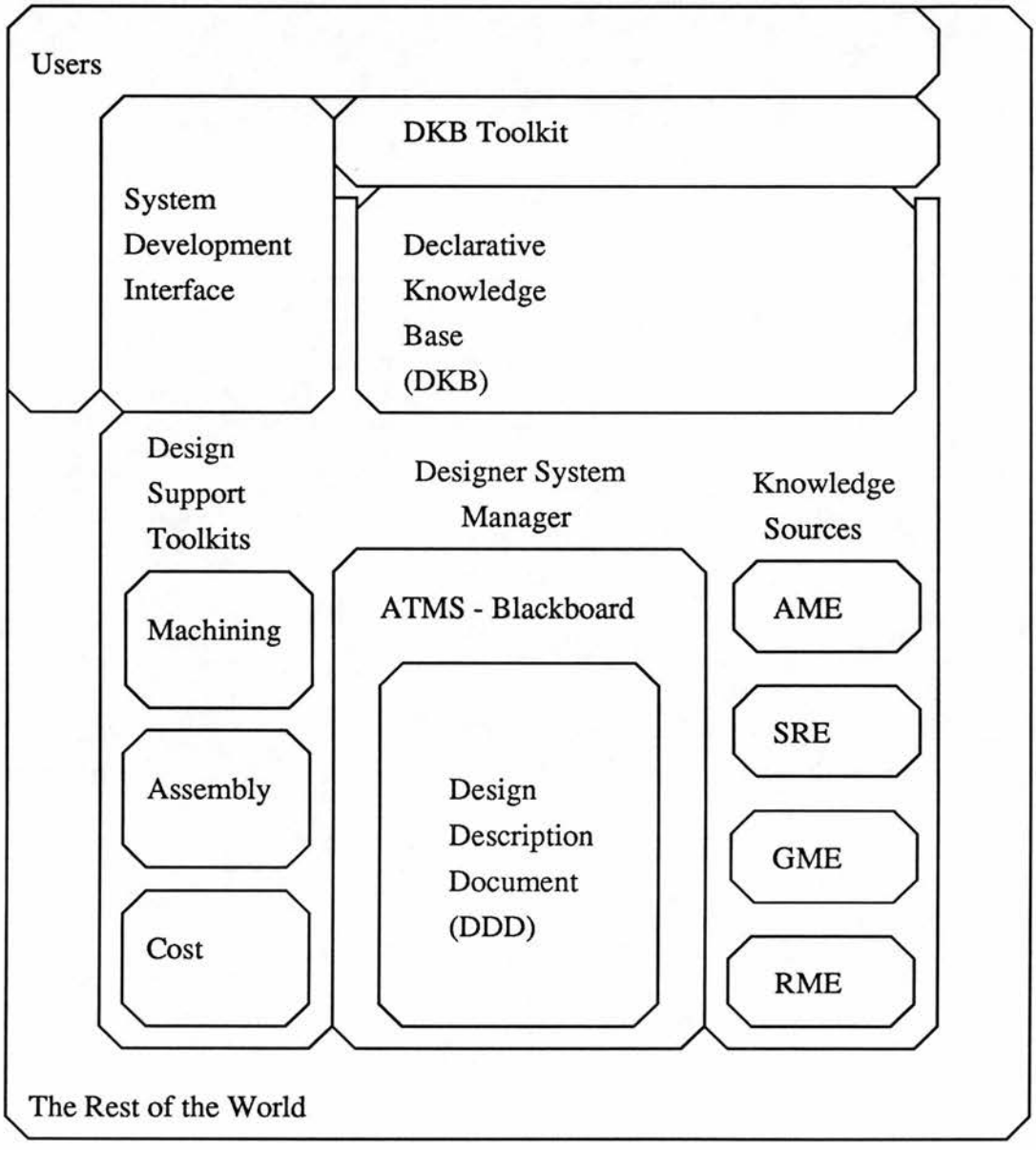


Figure 3–3: EDS Architecture

For mechanical engineering design, the Knowledge Sources include a Geometric Modelling Engine (GME) and a Spatial Relationship Engine (SRE) along with less domain-specific sub-systems such as an Algebraic Manipulation Engine (AME), a Relation Manipulation Engine (RME) and methods for constraint propagation. The Design Support Toolkits provide specific support for particular activities. For example, in mechanical engineering these activities might include: design of machining and design of robotic assembly. Currently, with the exception of a single test application, the Design Support Toolkits have not been implemented at Edinburgh.

The Knowledge Sources and Design Support Toolkits can be usefully distinguished by the *modality* of the inferences they make. The first produces necessary consequences whereas the latter offers possible consequences. Within this framework of automated reasoning, designers can build design descriptions from their assumptions and the consequences derived from them. What then is the role for the Declarative Knowledge Base (DKB) in supporting this process?

Declarative Knowledge Base

Designers use the DKB by making an assumption which creates an *instance* of a named packet of constraints represented in the DKB within the DDD. In the typically loose language of EDS users, this is called 'loading from' the DKB. Although the DKB provides a distinct mechanism for generating constraints within the DDD, the content of the constraints derived from assumptions drawing on the DKB does not differ from that which can be assumed by the designer directly. In other words, the influence of the DKB on the design process supported by EDS is not based on *what* knowledge resides in the DKB, but *how* this knowledge is structured and made available to the human designer.

The DKB uses a structured object representation scheme. The objects are implemented by Module Class Definition Files (MCDFs) created by the DKB-builder using a toolkit which includes utilities such as editing and syntax checking. These objects or MCDFs contain knowledge about the design space — both domain and

design knowledge — represented through a constraint-based language. Details of the MCDF structure are given in Appendix A.

Relationships between the objects in the DKB are limited to specialisation relations (with single parent inheritance only) called *kind-of* relations, and aggregation relations called *part-of* relations.

Figure 3–4 shows the specialisation relationship for a fragment of an example DKB. Note that the set of objects which is a *kind-of*, for example the three kinds of cONTACTbODY[2] are not exhaustive nor are they necessarily disjoint.

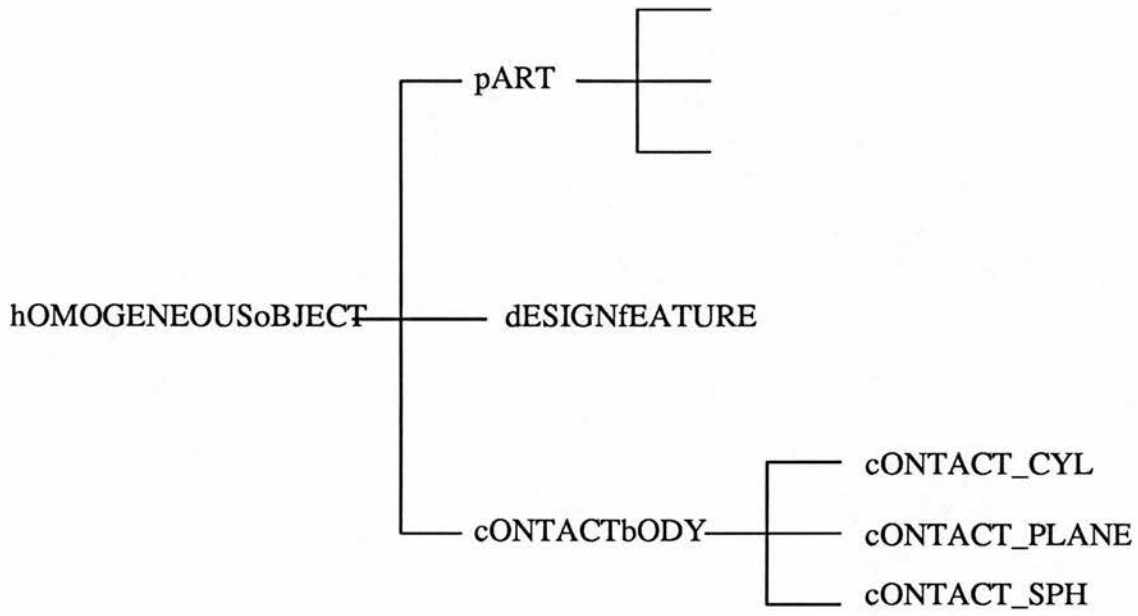


Figure 3–4: The Specialisation Relationship within the DKB

The aggregation relationship is shown in Figure 3–5. Note that aggregation is recursive in the sense that aggregation relationships may be defined in terms of

[2]The convention for using mixed upper and lower case letters in MCDF names is the author's own and not a feature or restriction of EDS. The aim is to clearly distinguish between potentially very different uses of the same word (For example, pART in Figure 3–4).

further aggregation relationships. For example, rOTOR is *part-of* pUMP which in turn is *part-of* fUELiNJECTIONsYSTEM. Whilst these relationships are known as *part-of* relations, there is no constraint that the objects so related are parts (in the engineering design sense). Assemblies of parts, design features or indeed any packets of constraints collected in distinct objects can be related by *part-of*. The generality of these structuring relations and the potential impact of this feature on DKB-building and the way designs are developed within product creation is discussed in Chapter 6.

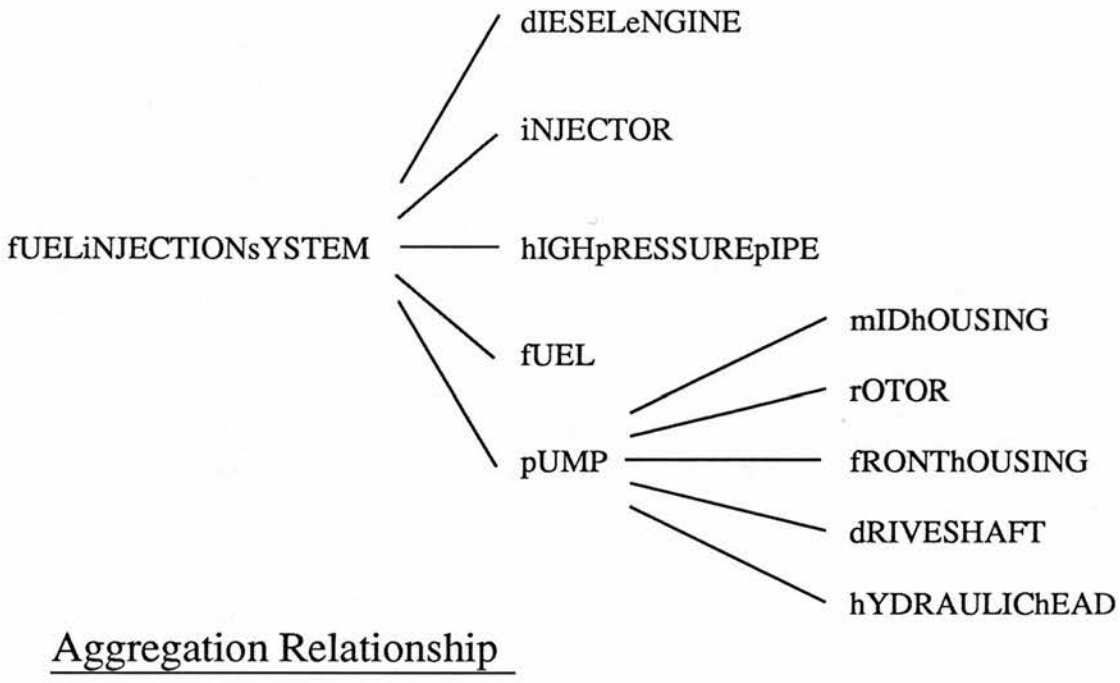


Figure 3–5: The Aggregation Relationship within the DKB

3.3.3 Solving Design Problems with EDS

In the previous section the EDS architecture was outlined. This section describes how this architecture and in particular the Declarative Knowledge Base (DKB) can be used by the human designer to support the solution of design problems. More specifically, the aim is to demonstrate two distinct, but related, ways in which EDS can be used to solve a single design analysis problem.

Objects are represented in the DKB using a constraint-based language. These objects can be compared to an implementation of the *prototypes* described in [Gero *et al*, 1988; Gero & Rosenman, 1990]. There are important distinctions between the MCDFs in the DKB and these prototypes. For example, prototypes represent domain knowledge such as requirements and parameterised design descriptions, but not design knowledge. The objects within the DKB are not constrained by this limitation and can be used to represent knowledge about how to design. Despite this, use of these object classes within EDS can be legitimately compared to the classification of the uses of prototypes as refinement, adaptation and generation proposed by Gero. (A discussion of the role of prototypes within the model of design presented in Figure 3–2 is given in [Logan & Smithers, 1989].)

Prototype refinement involves the instantiation of an object from its class and the assumption of values for the parameters of the object to form a design description. So-called 'routine design' is categorised as prototype refinement.

The second kind of prototype use corresponds, claims Gero, to innovative design and is called prototype adaptation. Prototype adaptation involves the modification of a prototype instance if it is found to be inadequate for the design task being tackled. In EDS, constraint-based object instances can, in addition to being modified through the assumption of extra constraints, be related to each other through constraints by the human designer. Thus, EDS can support users solving design problems in creative ways unforeseen by the DKB-builders without there having to be an existing (single) prototype for that problem and without the designer having to define their own new prototype. This type of design problem-solving behaviour is a form of prototype adaptation (though not one explicitly recognised by Gero) and constitutes a major way in which EDS can be used to support human designers[3].

---

[3]From the DKB-builder's point of view, the existence of this style of EDS use can inform the way in which the design and domain knowledge is structured and made available to the user. Note that to provide effective support it is not necessary to know *a priori* all the ways a designer would want to combine these packets of constraints.

The final type of prototype use is prototype generation. Gero claims that new prototypes are usually recognised after they have been generated and shown to be useful and that prototype generation is *"post facto recognition of a creative design"*. For EDS, prototype generation is an activity inextricably linked to DKB-building. That said, EDS can be used without reference to the DKB by assuming design constraints directly rather than via the constraint packets provided by objects in the DKB. In other words, EDS users can tackle problems for which there is no DKB support, but this is not easy. It is tempting to conclude, since DKB objects may subsequently be constructed to serve as prototypes for future occurrences of similar problems, that creative design has taken place. This is false. To conclude that is to confuse the role of prototypes and their implementation as objects in the EDS DKB. The lack of a suitable object in the DKB to assist the human designer with a specific problem indicates only one limitation of DKB coverage rather than suggesting that creative and not innovative or routine design is taking place.

The following design problem illustrates the differences between prototype refinement and adaptation, and the way EDS supports these activities. The example draws on a Design Drawing Office project undertaken by the author which in turn formed part of the application of EDS within the *Design-to-Product* project demonstration.

3.3.4 EDS Design Problem: Cam and Roller Analysis

Background

Diesel engines operate through the injection of fuel in controlled volumes at controlled times and at very high pressures, typically about 1000 bar. The fuel is raised to these injection pressures by a pump which compresses fuel by a plunger moved by the action of a roller on a cam. In practice, sets of constraints originating from commercial considerations or environmental regulations, for example, make the cam roller contact a crucial area of diesel fuel pump design. Applications of existing designs produce requirements to enlarge the plungers to improve the fuelling, increase pumping pressure to raise the rate of injection, or modify the

cam form to improve combustion and hence reduce harmful exhaust emissions. One key parameter is the contact stress on the cam form and the impact of this stress on the durability (in terms of a standardised measure of product life) of the pump.

Problem Statement

For reasons of brevity, the problem presented here has been simplified. A fuller treatment is given in Appendix A.

> *An existing product using standard parts is proposed for a new application requiring a higher pumping pressure and extended product life. Investigate whether the standard cam's life is acceptable. Propose design modifications if required.*

Prototype Refinement

Solving the problem using prototype refinement in EDS requires the existence of an MCDF whose parameters and constraints accurately reflect the problem space. In this case, the MCDF is called cAMrOLLERcONTACT. The constraints within the MCDF reflect the specific form in which more general equations are being applied. For example, the general equation for the Hertzian stress between two cylinders becomes (for a roller and cam with the dimensions shown in Figure 3–6):

$$S = 53284.57 d \sqrt{\frac{P(R_n + R_r)}{\cos\alpha L R_n R_r}}$$

In this equation, the force due to the pumping load, F_p in Figure 3–6, has been substituted for in terms of the cam stress, S, the area of a plunger of diameter d over which the pumping pressure P acts, and the angle α between the load force and the resultant force on the cam. The units adopted for the various parameters are typical for the domain rather than good practice. The 'constant' at the front of the right hand side includes both material constants and conversion factors between units! In design notes, such as [CAV, 1979], this specific form of

the contact stress equation is presented without explicit mention of underlying assumptions about the mechanical properties of the cam and roller materials[4].

Thus, given this specific form of the stress equation, to infer values for the cam stress (in kN/m^2) requires only the assumption of an instance of the cAMrOLLER-cONTACT MCDF from the DKB and the assumption of the known values for parameters. Note that if we are re-using existing parts, we need to assume values for the part numbers only and can make use of catalogue facilities within the MCDF. Only if the assumption of these basic values fails to produce the required values, or an inconsistency is derived, does the EDS user have to re-examine the structure of the design problem. In this case, for example, if a cam life value derived from the calculated cam stress is less than that required by the customer an inconsistency results. At this stage, some modification of the original design or some change in the requirements description is necessary. Note also how the 'compiled' nature of the design equations promote or preclude the modification of specific parameters. One solution would be to reduce the plunger diameter, thus reducing the stress and prolonging the cam life. Unfortunately, a possible consequence of this choice is that the modified design fails to meet the customer's fuelling requirement as a smaller diameter plunger pumps less fuel if the cam design is fixed. The fuelling system would thus becomes the next focus of the designer's attention.

Prototype Adaptation

Prototype adaptation is harder work for the EDS user than prototype refinement. The DKB does not contain a single MCDF which reflects the problem space. In this case, the solution is developed by recognising the similarity of the specific problem — the cam roller contact — to the general case of Hertzian stress between cylinders in contact. The DKB does provide some useful MCDFs for this kind of

---

[4]For example, this equation uses the modulus of elasticity and Poisson's ratio of mild steel for both the roller and the cam.

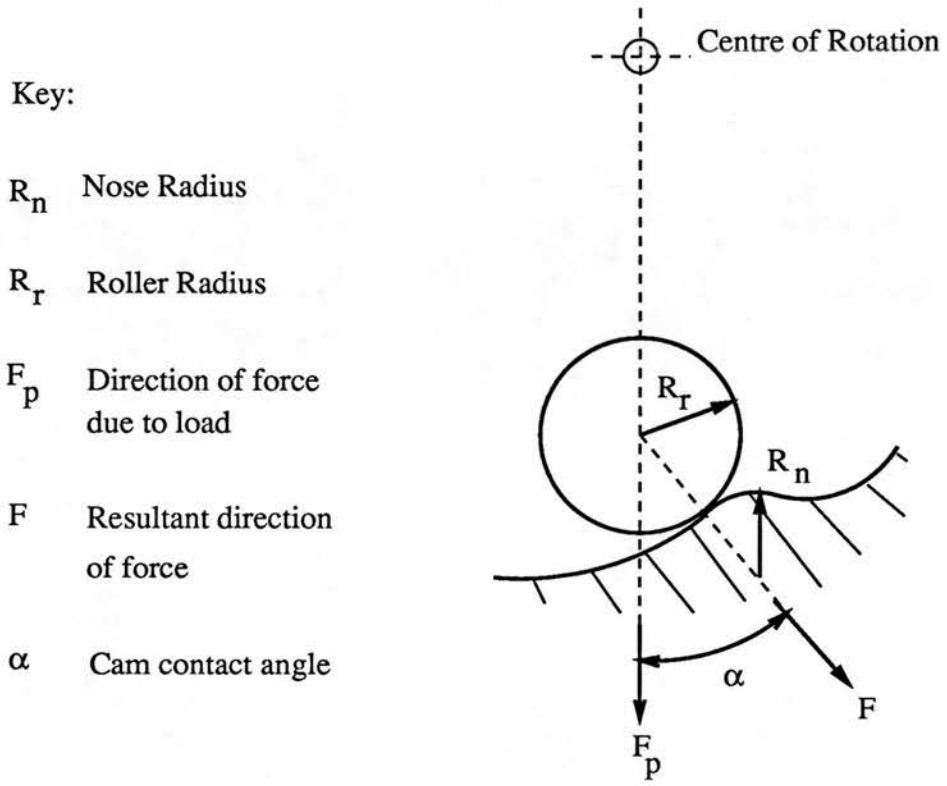


Figure 3–6: Internal Cam and Roller

problem and the solution begins by assuming an instance of the MCDF called cONTACT_CYLcYL. The general equation for deriving the contact stress is:

$$S = \frac{2F}{\pi BL}$$

Where F is the force on the contact area, L the length of the contact, and B, the half-width of the contact area. B is dependent on the geometry and material properties of the two cylinders in contact. In the notation of the cam roller example, the derivation of B requires values for the modulus of elasticity, E, and Poisson's ratio, ν, for the cam and roller along with the roller and cam nose radii, R_r and R_n.

The half-width of the contact area is given by:

$$B = \sqrt{\frac{2F}{\pi L}\frac{[(1-\nu_c^2)/E_c] + [(1-\nu_r^2)/E_r]}{(1/R_n) + (1/R_r)}}$$

This is the form given in standard engineering textbooks such as [Shigley, 1986]. The EDS user now has to adapt the general equation for the specific case of the

cam roller contact. For example, F in the equation is related to F_p and the angle α in Figure 3–6. The EDS user can do this by either finding additional useful constraint packets in the DKB or by making constraint assumptions directly.

To estimate the durability of the cam roller contact also requires the adaptation of another more general prototype. The designer can derive an estimate of cam life from a combination of the bearing life equation and empirical data. For the prototype refinement solution empirical data for existing designs was available within the MCDF and the life was derived directly as a consequence of the value for the contact stress. As with the prototype refinement solution – it is after all the same problem! – if the derived value for the cam life is less than that required by the customer an inconsistency results.

As before, some modification of the original design is required or some change in the requirements description. Note that the fuller structure of the design constraints in this case presents the specific parameters for modification in a different way. For example, material parameters are explicitly represented in the contact stress constraint. By assuming a reduced plunger diameter to lower the contact stress and prolong the cam life the designer can satisfy the durability requirement. In a more complex example, the fuelling reduction which is a consequence of reducing the plunger diameter may lead to further conflicts with the customer's requirements.

3.3.5 Tolerance Representation in EDS

The example in the previous section did not contain any tolerance statements. This is a serious deficiency of course and a weakness which is widespread in examples of problem-solving in AI-based design. Unlike systems based on nominal values, EDS does possess a tolerance statement construct within its constraint language.

In general, a tolerance constraint is of the form:

$$\text{tolerance}(tolerance\_type, toleranced\_expr) = tolerance\_expr$$

The *type* is limited to either angular or linear. The *toleranced_expr* may be either a single parameter or more generally an expression containing more than one parameter. The right hand side, the *tolerance_expr*, may take the same form as the *toleranced_expr*, but in addition can be the tolerance value. The following examples illustrate the kinds of tolerance statement possible:

$$\text{tolerance}(angular, alpha\$cam) = 0.2$$

$$\text{tolerance}(linear, c\$x) = a\$x + b\$x$$

Despite its admitted limitations there are several interesting points which arise from the provision of this tolerance representation capability.

Why EDS Needed Tolerance Statements

The arguments presented in Section 3.2 are *prima facie* evidence for the need to represent tolerance statements in AI-based design. The limited implementation in EDS has more pragmatic origins. Recall that work on EDS formed part of a project, *Design-to-Product*, which tackled the product creation process. In this project the design description developed within EDS acquired significance as a means of exchange of information between, for example, product design and process planning activities. In short, either EDS represented tolerances or the planners could not operate.

Inadequacies in Representation and Reasoning

Requirements for assembly planning and manufacturing planning were both considered when defining the tolerance constraint syntax. Of particular interest was the potential interaction between Edinburgh's work on EDS and Loughborough University's work on knowledge-based process planning (indeed their implementation followed Edinburgh's in being based around the use of an ATMS)[Hinde *et*

al, 1989; Herbert *et al*, 1990]. The following is a list of the main properties of the tolerance representation scheme adopted[5]:

- *Automated Reasoning about tolerance statements is unsupported.* No Knowledge Sources in EDS are able to derive consequences from the presence of tolerance statements in the DDD. Thus, none of the methods reviewed in Chapter 2 for representing valid variational models based on these tolerance statements are supported. Indeed, there are no facilities for combining tolerances on related parameters for simple cases, such as functional loops, where valid inferences can be made without reference to the variational model. The exception to this lack of reasoning support is *value propagation*, substituting values for parameters in expressions containing those parameters whose value is known.

- *Tolerance statements can be represented within the dependency structure of the DDD.* Functional requirements can be related to tolerance statements either directly or through value propagation. In other words, even this rudimentary syntax admits the use of variational along with nominal information in support of product creation.

- *Coverage of geometric tolerances is inadequate.* The EDS tolerance construct approximates to the conventional +/− tolerancing which was discussed with respect to Requicha's work in Chapter 2 where it was suggested that this form of tolerancing is insufficient and ambiguous. Geometric tolerancing in standard engineering form [BSI, 1984] requires reference geometric features in addition to dimensional values. These are not directly supported by EDS. In particular, datum lines and planes cannot be represented by EDS which uses a Geometric Modelling Engine based on Constructive Solid Geometry (CSG).

---

[5]The syntax discussed in this section was agreed at a meeting of the Loughborough and Edinburgh research teams. The resulting specification was clarified and implemented by Dr. Karl Millington at Edinburgh.

- *Tolerance statements can be related directly to geometry within the dependency structure of the DDD.* Despite the limitations of CSG for describing geometric tolerances, EDS users benefit from a constraint-based approach which allows a single parameter to be an argument within both a geometric constraint (a CSG expression) and a tolerance statement. Thus, dependencies between the human designer's assumptions about tolerance values and related geometric consequences in the DDD can be recorded and maintained by the ATMS. However, the dependency structure within the DDD relating the designer's assumptions (dimensions and tolerances) to the appropriate geometric constraints does not, of itself, describe a complete and valid variational model for the designed artefact geometry.

- *Incompleteness and redundancy in the tolerance specification are not defined.* As a result of the lack of automated reasoning about tolerance statements and inability to distinguish in EDS between a parameter of the geometric model and the designer's toleranced dimensions, a valid variational model of the designed artefact geometry cannot be developed. In practice, default values for linear and angular tolerances can be introduced to provide one type of completeness, but this guarantees that parameters are toleranced only and is insufficient to ensure that the consequent variational model is valid. In addition, EDS is unable to reason with these defaults and simply records them in the DDD.

In conclusion, there is some evidence that EDS and Design Support Systems more generally can provide a framework within which the pervasive nature of the tolerance statment can be represented. Little evidence has been presented so far to support the appropriateness of the EDS framework for the kinds of reasoning required with these tolerance statements and this representation.

3.4 Summary

This chapter has presented the problem scenario which this thesis addresses. In particular, the experimental objectives of the work presented here were derived from the motivating hypotheses. A strategy for exploring these objectives was outlined involving primarily a study of tolerance statements in AI-based design, but in addition investigating the way in which the AI-based approach to design is supported within product creation. A justification for studying tolerance statements in AI-based design was presented in Section 3.2.1.

The Edinburgh Designer System (EDS) was introduced as the apparatus to be used to pursue the experimental objectives. In addition to describing the architecture of the system and the design process model which underpins it, the way in which EDS is used to solve design problems was explained. The existing facilities for tolerance representation in EDS were discussed. The next chapter investigates tolerance combination reasoning and how this can be supported within EDS.

Chapter 4

Tolerance Combination

Tolerance combination is the process by which the effect of a number of tolerance constraints taken together is inferred. Specifically, tolerance combination techniques are used to infer the effect on a functional requirement of the set of tolerance statements made by the designer.

This chapter develops the description of tolerance combination introduced in Chapter 3. It is organised in three sections. In the first section, three tolerance combination techniques are presented: functional loop analysis, inference from degree-of-freedom constraints, and analysis of tolerance distribution. The consequences of trying to support these tolerance combination techniques within Design Support Systems is the subject of the second section. In particular, approaches to the implementation of these tolerance combination methods for the Edinburgh Designer System (an experimental DSS described in Chapter 3) are outlined. The chapter closes with a discussion of the general limitations of tolerance combination in terms of the support provided for human designers.

4.1 Tolerance Combination Techniques

4.1.1 Functional Loop Analysis

The notion of a functional loop is identified in the British Standards Institution guide to tolerancing functional dimensions [BSI Education, 1985]. A functional loop, drawn over the component geometry to which it relates, provides a visible representation of the constraint between a specific requirement and the nominal geometric embodiment which satisfies that requirement. In the spark plug example in Chapter 3, the requirement for the electrode air gap is represented with the surrounding component geometry (see Figure 3–1).

Functional loops are a technique for visualising tolerance constraints rather than for combining or allocating tolerance statements *per se*. The tolerance combination method actually employed is simple addition. Recall that the tolerance on the functional requirement is equal to the sum of the tolerance contributions within the functional loop. This is true for Figure 3–1 and for the examples in [BSI Education, 1985], but is not universally true as relationships between functional requirements and dimensions may be complex. Despite this, combination-by-addition is of widespread use even for apparently complicated designs. (A possible explanation for this is presented in Section 4.3.)

Functional loops and the combination-by-addition method are essentially one-dimensional. They can be extended to two and three dimensions by projecting the effect of any tolerance statement within the functional loop onto the co-ordinate direction of the requirement. Unfortunately, as the toleranced geometry becomes more complex, the simplicity of the combination method is offset by the additional effort required to transform the tolerance statements as given into the requirement functional loop representation.

The work of Ingham on tolerance combination develops the notion of the functional loop (though this is not the terminology used) into two dimensions [Ingham, 1980]. This work uses matrix multiplication rather than addition as the method

by which tolerance statements are combined. The method is in fact equivalent to projecting all tolerance statements onto a consistent pair of bases and solving for the two bases simultaneously. An important consequence of increasing the complexity of the analysis from one to two dimensions is that the effectiveness of a single tolerance statement in constraining a functional requirement depends both upon its magnitude and its direction. Thus, in two dimensions, and with the likelihood of several requirements being present, the tolerance combination activity can become increasingly difficult to control.

Figure 4–1 shows a triangular cover plate and the tolerances on one corner of this plate. Functional requirements constrain the positions of the three holes with respect to each other and the edge of the plate.

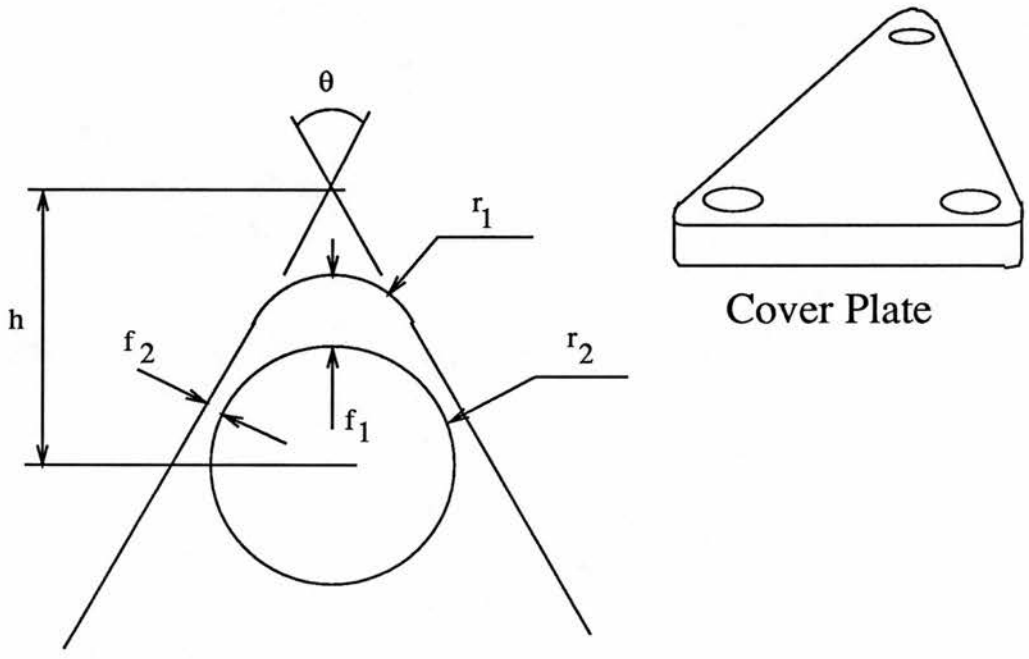
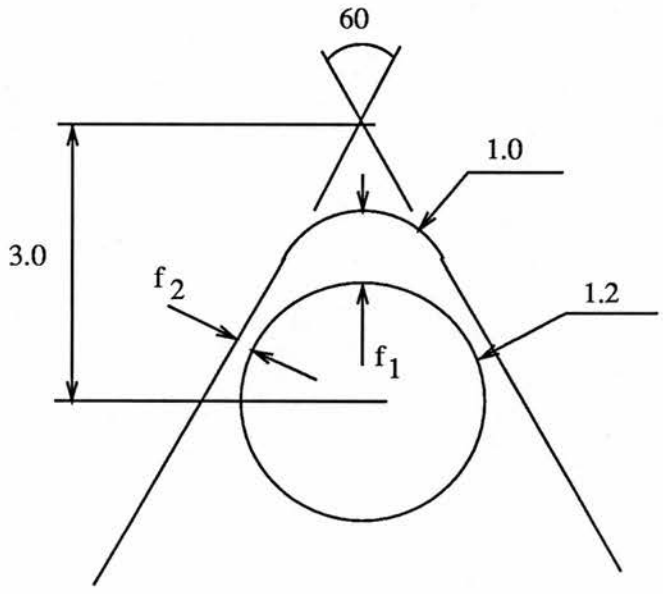


Figure 4–1: Tolerance Combination in Two Dimensions

In Figure 4–1, the two functional requirements labelled f_1 and f_2 represent the necessity for one hole to remain within a controlled distance of the edge of the plate. Four parameters of the plate corner are identified: r_1, r_2, h and θ. To determine how individual parameter tolerances combine we can project linear tolerances onto the directions of f_1 and f_2. More interesting and useful is to consider a single unit of tolerance acting at each of the parameters and determine

its effect on each of the functional requirements. In other words, to derive the sensitivity of the requirements to the individual parameters. Note that in this case the sensitivity of f_1 to θ depends on the interpretation of Figure 4–1. If r_1 is constrained to blend tangentially into the sides of the plate then changes in θ affect f_1. If this tangential condition does not apply then f_1 is independent of θ and the relevant sensitivity matrix element is zero[1].

A specific case of the geometric arrangement shown in Figure 4–1 and its corresponding sensitivity matrix are shown in Figure 4–2. Note that the elements of Ingham's sensitivity matrix are unsigned although the effects of, for example, h and r_2 on f_2 are clearly opposite.



	f_1	f_2
r_1 (1.0)	1.00	0.00
r_2 (1.2)	1.00	1.00
h (3.0)	1.00	0.50
θ (60)	1.26	2.60

Sensitivity Matrix

Figure 4–2: Example of a Sensitivity Matrix

Thus, the completion of the sensitivity matrix can serve as one approach to controlling the tolerance combination process for complex geometry requiring, as it does, the clear identification of functional requirements and contributing parameters.

---

[1]The author is grateful to Mike Cameron-Jones of the Department of Artificial Intelligence in Edinburgh for clarifying the alternative interpretations of Figure 4–1.

4.1.2 Inference from Degree-of-Freedom Constraints

Ingham's work on two-dimensional tolerance combination and the development of sensitivity matrices suggests a general approach to the combination of tolerance statements based on constraining each feature and its location relative to other features. Within this framework, functional requirements would be viewed only as the principal feature-to-feature relationships of interest to the designer. Such an approach, based on the maintenance of a tolerance network and the ability to derive subsequent tolerance combination inferences, is proposed in [Fleming, 1987].

Fleming adopts the tolerance representation scheme proposed by Requicha (reviewed in Chapter 2) in which complex geometric constraints are represented as a set of relations between tolerance zones and geometric datums. Constraints apply to each degree-of-freedom of each feature so that in two dimensions there are three relations whilst in three dimensions six relations are required.

In general each constraint is of the form:

$$expr_1 \leq \text{dof} \leq expr_2 \tag{4.1}$$

Where $expr_1$ and $expr_2$ are the two expressions bounding the degree-of-freedom (dof). For the case where tolerances are applied, each degree-of-freedom constraint gives rise to two others representing the bounds on the degree-of-freedom variable and the limits on its variation. Even in two dimensions with degrees-of-freedom x, y and θ this constraint expression formulation appears somewhat convoluted: its value arises from its generality. For example, individual features, individual parts or collections of parts can all be represented. In the last case, that of assemblies of parts, bounds represent the limits of variation within the assembly (what Fleming calls *"slop"*).

For the example part introduced in Figure 4–1, the functional requirement f_1 can be framed as constraints on a degree-of-freedom y. In the simplified case where variations in θ are neglected, for example, if the tangency constraint on r_1 does

not apply, the set of constraints on the parameters of the requirement is shown in Equations 4.2 to 4.4:

$$h + tl_h \leq h \leq h + tu_h \tag{4.2}$$

$$r_1 + tl_{r_1} \leq r_1 \leq r_1 + tu_{r_1} \tag{4.3}$$

$$r_2 + tl_{r_2} \leq r_2 \leq r_2 + tu_{r_2} \tag{4.4}$$

(Where tl and tu represent the permitted variations below and above the nominal. For conventional +/– tolerancing practice $tu - tl$ represents the total tolerance and $tl = -tu$.)

Thus, for the simplified version of the functional requirement f_1 in Figure 4–1, the bounds on degree-of-freedom y are given by:

$$(h+tl_h)-(r_1+tu_{r_1}+r_2+tu_{r_2}) \leq yu-yl \leq (h+tu_h)-(r_1+tl_{r_1}+r_2+tl_{r_2}) \tag{4.5}$$

Equation 4.5 should by now be familiar — it is a re-arranged version of the equation defining the variational form of a functional loop introduced in Chapter 3! Beyond the generality of the representation, the utility of the Fleming approach lies in the use of *bounding algorithms* to find the limits for degree-of-freedom variables in the cases where complex sets of constraints are present. Fleming's bounding algorithms are based on the SUPINF method developed by Brooks [Brooks, 1981]. Thus, Fleming's work supports the derivation for a designed assembly of the bounds on individual part motions. What the work is not intended to address is how the design assembly tolerances are allocated (the subject of Chapter 5) and how, given bounds on some part motion, we infer the *likelihood* of a particular instance of the designed assembly functioning correctly. In other words, the way instance values are distributed within tolerance bounds and the consequences of these distributions on how tolerances combine.

4.1.3 Analysis of Tolerance Distribution

Any dimension of a real designed artefact will vary in its value between instances of that designed artefact. Not only do real manufacturing processes necessarily introduce such variations, but in practice inspection methods place finite bounds on our ability to measure and control this variability [2].

By allocating tolerances, designers aim to ensure that instances of designed artefacts will function satisfactorily despite inevitable dimensional variability. That said, if each dimension is subject to variation from its nominal value, how can we analyse the likelihood of a particular instance failing to satisfy the dimensional tolerance? More importantly, in real designs where complex functions are realised by constraints between toleranced dimensions, how is the likelihood of satisfying these ultimate functional requirements related to the underlying dimensional variability?

In Parkinson's work on tolerance distribution [Parkinson, 1984], the Gaussian, or normal distribution is taken as the basic model of the relationship between a dimension x and the probability $p(x)$ of an instance of that dimension having a particular value (see Figure 4–3 (a))[3]. The adoption of Gaussian models for manufacturing process variability serves as a basis for developing process capability measures: comparing a sample mean ($\bar{x}$) and variance (σ^2) of a dimension x with the design specification's nominal and tolerance values.

The basic Gaussian model can be enhanced for the case where inspection pro-

---

[2]Not all design parameters are subject to this type of variation. For example, we would expect to control and verify the number of teeth on a gear precisely.

[3]More formally, $p(x)$ is the probability density function for the dimension x and the probability of a dimension instance having an *exact* value of x is zero.

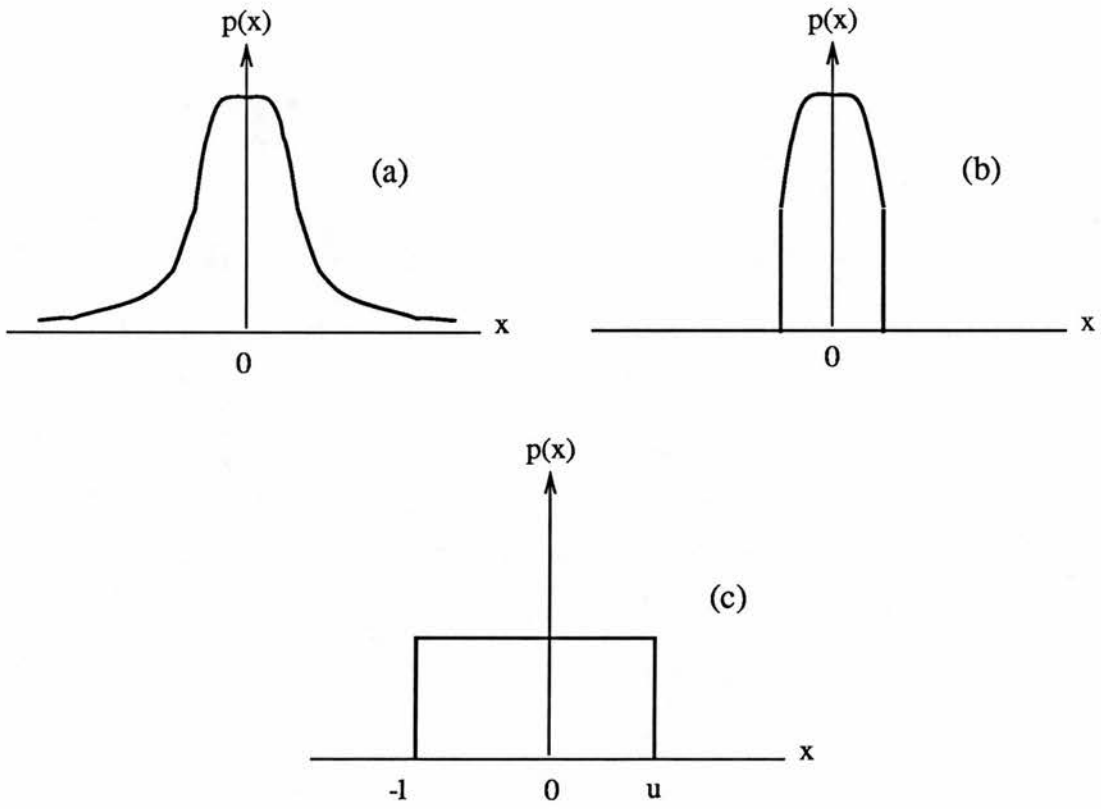


Figure 4–3: Statistical Models of Manufacturing Variability

cesses remove instances outside upper and lower tolerance limits by using a truncated Gaussian model as shown in Figure 4–3 (b)[4]:

Other statistical process models are possible. For example, the manufacture of a series of graded spacers or shims might approximate to a uniform distribution where the upper and lower dimensional bounds (u and l) represent the constraints on a particular shim or spacer grade. This situation is shown in Figure 4–3 (c).

For a functional requirement r, the probability $\mathrm{p}(r)$ that an instance of a designed artefact satisfies r is governed by the individual contributing probabilities $\mathrm{p}(r_1)$, $\mathrm{p}(r_2)\ldots\mathrm{p}(r_n)$ and the relationship between the individual requirement variables r_1 to r_n. Two simple cases are of particular importance:

Multiple Constraints on a Single Variable

When more than one constraint on a single variable is present, the resultant probability is found from the intersection of the bounds on that variable. Non-intersection of bounds, $\mathrm{p}(r) = 0$, can be interpreted as constraint conflict. The subsumption of one constraint by another can be interpreted as redundant toleranicing of that requirement variable.

Constraints Between Independent Variables

When individual requirement variables are independent of each other, the probability of satisfying r, $\mathrm{p}(r)$, is given by the product of the contributing probabilities $\mathrm{p}(r_1)$, $\mathrm{p}(r_2)\ldots\mathrm{p}(r_n)$. Thus for 2 independent variables we have:

$$\mathrm{p}(r_1 \wedge r_2) = \mathrm{p}(r_1)\mathrm{p}(r_2) \tag{4.6}$$

Although these two special cases are widely applicable, the general relationship between variables r_1, $r_2 \ldots r_n$ is more complex and some measure of relatedness of

---

[4]Parkinson claims that the truncated normal distribution can be transformed and characterised as a normal distribution with modified parameters, $\bar{x}_t$ and σ_t^2.

the variables to each other is required. These correlation metrics can be derived by statistical analysis of the functional requirements and their contributing variable distributions. This analysis is complex and beyond the scope of the thesis: details and a worked example are included in [Parkinson, 1984].

4.2 Supporting Tolerance Combination in Design Support Systems

In the previous section a number of different approaches to the tolerance combination activity were presented. Methods for drawing tolerance inferences included: simple addition, matrix multiplication, the use of bounding algorithms, and statistical analysis. This section discusses the implementation of these methods within Design Support Systems (DSSs) as characterised in Chapter 2. Where possible, the Edinburgh Designer System, an experimental DSS, and its limited tolerance representation capabilities are used to illustrate the practical constraints on supporting tolerance combination.

4.2.1 Functional Loop Analysis

For a simple functional loop, of the kind introduced at the end of Chapter 3, implementing tolerance combination support requires only the specification of the variational form of the functional requirement and the availability of an equation solver. The equation solver itself must support the substitution of known values into expressions and the derivation of single unknowns from these expressions. Thus, for the Edinburgh Designer System implementing this tolerance combination support requires:

- Existence of a node within the Design Description Document (DDD) which contains the variational form of the functional requirement. For example:

```
tolerance(linear,f$p) = tolerance(linear, z$p)
                      + tolerance(linear, x$p)
                      + tolerance(linear, y$p)
```

- Existence of nodes within the DDD containing values for the tolerances already allocated. For example:
  `tolerance(linear, x$p) = 0.4`

- Inference methods for tolerance combination using nodes within the DDD. For example, in EDS *valsInToleranceExpr* for propagating values into tolerance expressions and *valsInMulti* for propagating values from multivariate expressions such as functional requirements.

To illustrate how EDS capabilities can be used to support tolerance combination by addition within a functional loop, consider the spark plug example shown in Figure 3–1. A cut-down version of a Module Class Definition File (MCDF) which might be used to support the design of petrol engine spark plugs is shown in Figure 4–4. (An example of a full MCDF listing is given in Appendix A.)

For an instance labelled p of the pLUG MCDF, the tolerance values for fp, yp and x$p might be derived from the requirements statement, tolerance allocation from a standard component, and a designer assumption respectively. Given tolerance values within the DDD for fp, yp and x$p, the value of z$p is inferred from the instantiated form of the variational constraint labelled #2## in Figure 4–4. A simplified fragment of the DDD representing these assumptions and inferences is shown in Figure 4–5.

Although Figure 4–5 clearly demonstrates how EDS is capable of supporting tolerance combination, there are two limitations within the current implementation:

```
    class name [X]: pLUG

beginParameters

    f$X      real    mm       #1 Air gap: functional requirement ##
    x$X      real    mm       #2 Exposed core electrode ##
    y$X      real    mm       #3 Exposed core insulator ##
    z$X      real    mm       #4 Base electrode clearance ##

endParameters

beginSection: constraints

    f$X = z$X - y$X - x$X                                  #1 ##

    tolerance(linear, f$X) =  tolerance(linear, z$X)
                            + tolerance(linear, y$X)
                            + tolerance(linear, x$X)   #2 ##
endSection: constraints
```

Figure 4–4: MCDF for Tolerance Combination

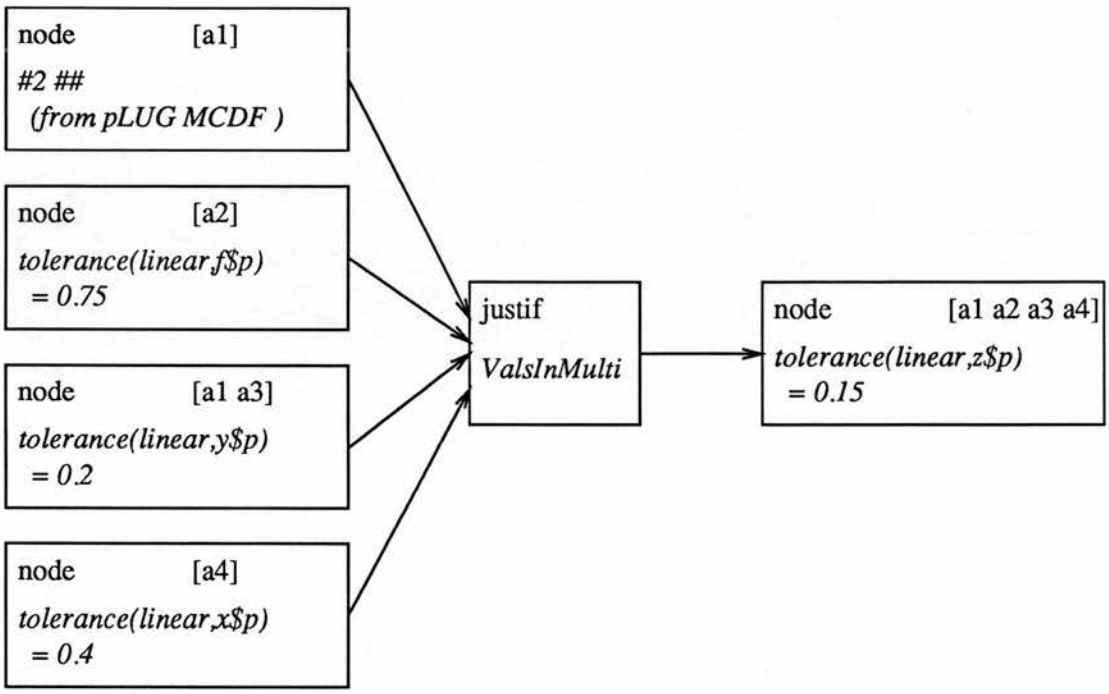


Figure 4–5: Design Description Document for Tolerance Combination

- *There are no EDS methods which support the derivation of the variational form of the functional loop from its nominal statement.*
  The two constraints in the pLUG MCDF of Figure 4–4 must both be stated explicitly although in principle the second is derivable from the first given an adequate tolerance representation within the constraint language.

- *EDS has no explicit representation of the functional requirement.*
  Each parameter of the functional loop constraint is treated equally. Clearly in the context of the design exploration fragment represented by Figure 4–5 each tolerance value has its own distinct and different history. For example, the tolerance on f$plug arises from the requirements statement (labelled a2) whilst the tolerance on y$p might be based on the assumption of a standard component type (a3) given catalogue constraints within the MCDF (a1). Thus, any special meaning attached to the functional requirement, f$p, by the designer is not shared by EDS in its current implementation. Note however that the assumption bases within the DDD, as shown in Figure 4–5, could form the starting point for such distinctions.

These limitations would not prevent EDS capabilities for functional loop analysis being extended to include tolerance combination by matrix multiplication (as described in Section 4.1.1). Currently, EDS supports neither the projection of requirements onto suitable bases nor the tolerance multiplication method itself. Specific problems such as that in Figure 4–1 can be represented within the constraint language of the MCDF, but clearly the DKB-builder is required to invest effort in transforming tolerance statements into an essentially 1-dimensional form.

Implementing methods for deriving sensitivity matrices within EDS is hard. Deriving sensitivity matrices requires automated reasoning with both the functional loop constraints and the geometric constraints within a spatial occupancy model of the design. The Geometric Modelling Engine of EDS uses design parameters to instantiate Constructive Solid Geometry only. Thus, the underlying geometric relations of the spatial occupancy model are not accessible and so cannot contribute to the determination of sensitivity matrix coefficients. For this reason, the implementation of this kind of tolerance combination support within EDS has not been attempted.

In summary, DSSs and specifically EDS are capable of supporting functional loop analysis by simple addition and in principle by matrix multiplication (though this has not been demonstrated). The explicit construction of sensitivity matrices has not been shown[5]. Another deficiency which has not been addressed is how to support functional loops as a visualisation tool: we have said nothing about how the constraints and methods described here are presented to the user. This is one of the issues discussed in Chapter 7.

4.2.2 Inference from Degree-of-Freedom Constraints

The implementation of bounding algorithms to combine tolerances on degrees-of-freedom within EDS has not been attempted. This section examines the obstacles

---

[5]If we exclude Ingham's original implementation which was outside a design support framework.

to developing these implementations and argues on the basis of experience with EDS that these obstacles can be overcome.

Fleming's work on geometric tolerances is related to earlier work on spatial relationship inference [Corner *et al*, 1983], but does not make use of geometric models of the features for which tolerance inferences are drawn. This is in contrast to the implementation used to test the tolerance representation scheme developed by Requicha, and adopted by Fleming, which explicitly relates each variational feature to a nominal feature of a geometric model [Requicha & Chan, 1985]. An experimental implementation of bounding algorithms within EDS should employ both the geometric and spatial reasoning systems (known as GME and SRE respectively) available within its DSS framework.

There are two problems underlying the implementation of tolerance combination support based on degree-of-freedom constraints:

How are the degree-of-freedom expressions built-up?

Each feature in three dimensions gives rise to six degree-of-freedom constraints each of which may be interpreted as a set of three inequalities for the case where tolerances are applied [Fleming, 1987]. In engineering design the initial requirements statement does not typically provide the detail necessary to construct a usable degree-of-freedom network. Thus, whilst tolerance combination can be used to analyse structures of parts *which are already designed*, we need to support the construction of degree-of-freedom constraint sets as new design specifications are developed.

One approach to overcoming this deficiency within EDS is to attach degree-of-freedom constraints to spatial relationship features used by SRE and to associate these features with primitive surfaces accessible to the GME. This mechanism, combining elements of Fleming's and Requicha's experimental implementations, appears to provide a means whereby degree-of-freedom expressions can be constructed in EDS either directly or via spatial relationship or geometric assumptions.

What inferences are required and how are they invoked?

The generality of representation and reasoning made possible by degree-of-freedom expressions means that we can in principle determine the bounds on a degree-of-freedom of any feature in a complex assembly. The complete enumeration of all bounds would be computationally expensive and involve a considerable amount of unproductive effort. Individual degrees-of-freedom differ markedly in their contribution to satisfying the requirements motivating the design process. For experimental implementations to be tractable these distinctions need to be recognised and used.

This is the problem of providing intelligent control of inferencing methods. One simple strategy for degree-of-freedom inference is to rely upon user-invocation. Within a DSS framework this is an entirely legitimate approach[6]. Experience with the Spatial Relationship Engine (SRE) in EDS which is also user-invoked suggests that relying on the user alone is indeed a practical solution. Designers can identify the few spatial relationships of interest from the many potentially derivable. Unfortunately, in the current EDS implementation SRE is unable to make use of these user distinctions.

4.2.3 Analysis of Tolerance Distribution

Implementing tolerance combination support which includes statistical analysis within a DSS framework requires:

- *Design descriptions which can express the characteristics of parameter value distributions.*

  For example, along with an assumed value and allocated tolerance, we need to be able to specify that param$instance has a Gaussian distribution with a mean equal to its assumed nominal value and a given standard deviation.

---

[6]This option is not one open to AI in Design researchers who aim to automate engineering design tasks.

- *Reasoning support systems capable of deriving tolerance distribution inferences.*

  One simple example would be the derivation of the fraction of instances of a designed artefact which would fail to satisfy a known parameter tolerance given a characterisation of the distribution of the instance values.

  Another simple example of general applicability is the failure of a peg-in-hole assembly operation. Given distribution characterisations for the two designed artefacts and assuming that the parameter values can be treated as independent variables, Equation 4.6 can be used to infer the fraction of instances which will not assemble. In general, increasingly complex statistical analysis based on the combination of distributions, as introduced in Section 4.1.3, is required to determine these failure-to-assemble fractions where functional requirements involve many related parameters.

- *Control mechanisms which permit designers to use tolerance distribution analysis effectively.*

  For complex tasks like distribution analysis, DSSs need to support the user's construction of constraint sets to which the desired inference methods can be applied. Experience with the Spatial Relationship Engine in EDS suggests that designers know when inference methods should be invoked, but need help in collecting the antecedents necessary for their successful activation.

Currently, EDS is not capable of supporting the analysis of tolerance distribution. The obstacle to providing this facility is the requirement for major extensions to EDS design description capabilities. Whilst we can easily propose constructs for tolerance distribution representation along similar lines to the quality loss function constructs which will be presented in Chapter 5, two observations suggest we should be cautious in attempting to extend EDS capabilities in this direction:

Referring to Instances of the Designed Artefact

The tolerance construct of Chapter 3 and the quality loss function construct proposed in Chapter 5 both represent attempts to attach properties to design param-

eter instances. Characterisations of parameter instance populations also fall into this category, so we might have:

```
gauss(xbar, p$i) = 2.5
gauss(sigma, p$i) = 0.4
```

EDS design descriptions refer to an instance label: the instances of the designed artefacts themselves are not part of the EDS design language. As EDS users we cannot infer conditional probabilities such as the fraction of pegs that will not assemble given an instance of hole artefact with a specific parameter value. There is no way of referring to instances of the design instance.

Probabilistic analysis of design descriptions leads inevitably to the need for constraint languages which allow and make clear the distinction between individual instances and characterisations of populations.

Supporting Distribution Assumptions

If the application of Assumption-based Truth Maintenance Systems to design descriptions is to be successful, careful consideration is required as to what constitutes an admissible assumption. In particular, assumptions about the way a parameter's instance values are distributed represent the consequences of reasoning by the designer outwith the truth maintenance system in domains connected to the production rather than specification of designed artefacts. Of particular concern is the possibility that misconceptions in design about available manufacturing technology generate assumptions which over-constrain the design description and undermine the advantages of basing the product creation process on a truth-maintained description.

4.3 Discussion

Each of the methods described in Section 4.1 has practical limitations some of which are apparent from our attempts to support tolerance combination in Design Support Systems (Section 4.2). One shortcoming is the number of antecedents attached to each tolerance combination consequence. For example, to determine the tolerance distribution of a functional requirement might require complex statistical analysis based on assumptions that all contributing variables are normally distributed with known (or guessed) values for each distribution's mean and variance.

To minimise these prerequisites and simplify the derivation of tolerance inferences designers typically confine their attention to 1-dimensional functional loops and work with parameter limit values not inequalities and distributions. Current CAD systems provide a limited form of automated reasoning for this activity: given a user-identified functional loop the CAD system will compute the tolerance limits on the requirement given known contributing dimensions. The simplification to one dimension proves surprisingly effective. One explanation for this is suggested by Pahl and Beitz. Their guidelines to embodiment design stress the importance of *"short and direct force transmission paths"* and the avoidance of *"sharp deflections of the flowlines of force"* [Pahl & Beitz, 1988]. Such force transmission conditions would typically result in a requirement represented by a 1-dimensional functional loop.

Inevitably some assumptions have to be made before tolerance combination inferences can be drawn, and the process of making these antecedent assumptions, tolerance allocation, is unsupported by the methods described in this chapter. One could envisage the repeated use of tolerance combination techniques to direct these tolerance value assumptions (a kind of iterative "what-if"), but for many design tasks this generate-and-test approach is inadequate: we require additional tools to support tolerance allocation.

Chapter 5

Tolerance Allocation

Tolerance allocation is the process of assigning tolerance statements during design. These assignments are either decisions made directly by the designer or consequences of the requirements description and the designer's other choices. Thus, tolerance combination, the subject of Chapter 4, may be viewed as a very restricted kind of allocation: a single tolerance assignment being inferred from a combination of other known assignments.

This chapter addresses the general task of supporting the tolerance allocation process. The chapter is in three sections, in the first section two approaches to tolerance allocation are described: the use of standards and experience for tolerance assignment being contrasted with the systematic use of a quality loss function. The second section describes the consequences of trying to provide tolerance allocation techniques within Design Support Systems. Specifically, approaches to the implementation of these tolerance allocation methods for the Edinburgh Designer System (an experimental DSS described in Chapter 3) are outlined. The final section critically discusses the feasibility of providing tolerance allocation support and questions whether a complete solution is an appropriate one.

5.1 Tolerance Allocation Techniques

5.1.1 Standards and Experience

In the previous chapter, we saw how simple addition could be used to combine tolerance statements which had been allocated within a functional loop. Missing from the description was an explanation of how the tolerance statements to be combined were initially allocated. Examination of the eight examples in the British Standards Institution guide to tolerancing functional dimensions [BSI Education, 1985] provides the following characterisation of the tolerance allocation activity:

Characterisation of Tolerance Allocation

- Tolerance allocation begins with a functional requirement stated in terms of bounds on satisfactory performance. For example, for a relief valve, bounds arise from operating at the limits of the open and closed positions. Note that the functional requirement itself is not toleranced in that no additional constraints are placed on the functional requirement. This type of constraint can arise and is discussed with respect to the loss function in Section 5.1.2.

- Use of existing designed artefacts or standard parts, such as nuts and washers, introduces tolerance statements which have been previously allocated. Thus, for a given assembly of standard parts, tolerance combination can determine whether functional requirements are satisfied: tolerance allocation is implicit.

- Where limits on a functional requirement are known and standard parts are not used, practical tolerance allocation methods can be very simplistic. For example, one approach divides the functional tolerance equally amongst the contributing functional dimensions. An alternative method allocates wider tolerance bounds to larger dimensions.

- Tolerance allocation reflects *"previous experience"*, *"relative manufacturing difficulties"* (for the contributing dimensions) and which dimension's control would be *"most difficult to achieve"* [BSI Education, 1985]. Thus, for known functional requirement bounds and where standard parts are not involved, tolerance allocation can depend on activities outside the immediate design exploration process.

In summary, tolerances can be allocated by: selecting standard components with known tolerances, adopting simple allocation methods to meet a requirement, or using knowledge of activities such as manufacturing. Another approach based on the development of allocation methods which explicitly uses parameters outside design, for example manufacturing rework costs, is introduced in Section 5.1.2.

Standard Components

Specifications of engineering components which are in widespread use (examples include bolts and plain bearings) are published by standards organisations such as the British Standards Institution. These design specifications include tolerance statements for the dimensional parameters which are directed at satisfying typical functional requirements for that kind of component. Hence a standard washer with the required nominal geometry might not specify an appropriate tolerance for a design application where the washer was acting to provide dimensional control (as a shim) as well as distribute a load. (The use of standards in engineering design is discussed in [Pahl & Beitz, 1988].)

In addition to specifying components, standards are used to allocate tolerances to provide particular functionality. The principal example of this is the allocation of tolerances to shafts and holes to provide specific fits over a range of nominal hole sizes. Thus, to allocate tolerances to shafts or holes requires a basic size for the shaft or hole and a functional description of the kind of fit: clearance, transition or interference. Nominal sizes for both the hole and shaft and appropriate tolerances can then be derived directly from the standard (resulting in a fit definition as shown in Figure 5–1).

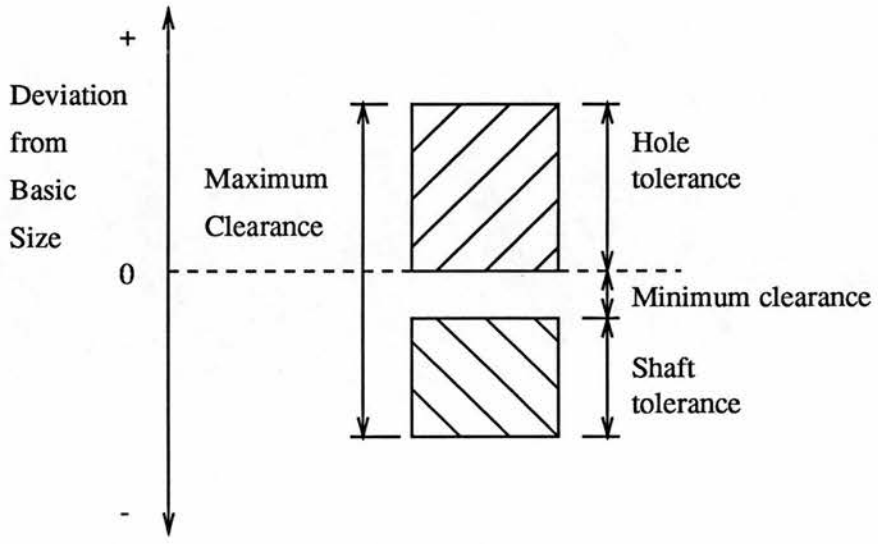


Figure 5–1: Defining a Clearance Fit

In other words, the justification of the specification of a particular fit in terms of its parameters is based on the designer's assumption of the functional description and the derivation of values from a defined standard.

Experience

Engineering standards set down good practice. Standards are developed by both theoreticians and practitioners, but their value is founded on and enhanced by successful adoption and use to develop real design specifications. Thus, an accepted standard represents good practice through the successful design exploration experiences to which it has contributed.

As well as the formal and collective representation of experience as standards, tolerance allocation is based on an individual designer's own experience. A designer can use their experience of previous design explorations to assume values and justify those assumptions during the tolerance allocation process. The following two examples illustrate this:

1. When modifying an existing design specification, characterised as prototype refinement in Chapter 3, a designer might 'carry forward' tolerance allo-

cations from the superseded design description justifying this assumption through reference to an underlying invariant functional requirement.

2. Tolerance allocations during the development of a new design might assume the process capabilities of an existing manufacturing facility. For example, a designer can use experience to avoid reproducing manufacturing difficulties which surfaced with previous designs.

In contrast to Rehg *et al*'s critique of tolerance allocation as typically determined by *"tradition, trial and error, or intuition"* [Rehg *et al*, 1988], it is clear that the adoption of standards and application of experience provides a valid framework for developing successful design descriptions. That said, we need to examine the alternative framework proposed in [Rehg *et al*, 1988] for the *"rational choice of tolerances"*: systematic methods for tolerance allocation. Chief among the measures intended to support this rational tolerance allocation is the quality loss function.

5.1.2 Quality Loss Function

The quality loss function is the name given to the quadratic approximation for the relationship between a characteristic's worth and the deviation of the characteristic from its nominal value. In its simplest form, the quality loss function $\mathrm{L}(y)$ of a characteristic y with nominal value m is given by:

$$\mathrm{L}(y) = \frac{\mathrm{A}}{\Delta^2}(y - m)^2 \tag{5.1}$$

Where A is the quality loss known to be associated with a characteristic deviation Δ. The derivation of Equation 5.1 is included in this thesis as Appendix B.

Despite its apparent simplicity, the quality loss function can be applied in a number of ways to support the tolerance allocation activity.

Tolerancing a Functional Requirement

Tolerance values on functional requirements are typically known and given. The tolerance allocation activity aims to ensure that these known functional requirement constraints are satisfied. Perhaps surprisingly, the loss function can support the derivation of additional bounds on the functional requirement as part of the tolerance allocation process.

For example, where costs associated with replacing defective components of a designed artefact during initial assembly are much lower than ultimate replacement "in the field", the adoption of tighter constraints on functional requirements in-house will be justified.

Tolerance Design

The loss function can be used as a method for inferring tolerance values for design parameters: a process which Taguchi calls *tolerance design* [Taguchi, 1986; Taguchi *et al*, 1989]. Based on assumptions of the costs associated with deviations of nominal parameter values, tolerance values which minimise the sum of the product cost and quality loss are derived. This process is a restatement of how functional requirements can be toleranced, in that the results of the tolerance design in fact represent additional constraints within bounds defined by functional limitations.

Inspection and the Loss Function

The loss function can be used to compare the costs of inspecting an artefact to ensure that a functional requirement is satisfied with the costs arising from accepting that a proportion of uninspected artefact instances will fail that requirement. Use of this technique as part of the tolerance allocation activity requires a set of tolerance values to serve as reference deviations against which costs can be associated. In addition, explicit assumptions about the expected process capabilities and inspection strategies to be employed during the production of the designed artefact are necessary. In other words, the loss function is used to derive the inspection

cost consequences of assumed tolerance values and thus informs the allocation of those values.

Each of these methods for supporting tolerance allocation relies on considerable knowledge of costs associated with deviating from nominal parameter values and knowledge must in turn be based on further assumptions. For example, Taguchi's tolerance design method suggests that the tolerance value for a given component parameter depends upon the cost of the material from which the component is to be manufactured. Section 5.3 critically discusses both the validity of the antecedents required to support this tolerance design and the legitimacy of the process itself.

5.2 Supporting Tolerance Allocation in Design Support Systems

5.2.1 Standards and Experience

Specifications for standard components and tolerances for satisfying known functional requirements can readily be utilised within design support systems. This is possible provided that the knowledge representation scheme supports constraints in tabular form and that the automated reasoning capabilities permit the manipulation of, and the derivation of values from, those tables. The Edinburgh Designer System (EDS) through its Relationship Manipulation Engine (RME) meets this prerequisite, and the following examples illustrate how standard components and known functional tolerances can be used to support tolerance allocation.

Standard Components and Prototype Refinement

Chapter 3 describes an example of the use of EDS to solve design problems through the process characterised by Gero as prototype refinement [Gero *et al*, 1988; Gero & Rosenman, 1990]. That example made use of a single Module Class Definition

File (MCDF) within the EDS Declarative Knowledge Base called cAMrOLLER-cONTACT. In that MCDF (listed in Appendix A) standard parts are represented by a constraint of type *catalogueTable*. Although tolerances are not allocated as part of the assumption of cam type in that example, the *catalogueTable* constraint can be extended to support this.

For example, consider a roller component with parameters length, l, and diameter, d. In the rOLLER MCDF, standard components of this class and their dimensional tolerances can be represented. Figure 5–2 shows, for a typical name variable R, the required constraints:

```
beginSection: constraints

catalogueTable  Standard Roller Dimensions  ##

rOLLER_TYPE$R          d$R                    l$R

typeA                  5.50                   12.00
typeB                  9.00                   17.50
                                                          #1 ##
catalogueTable  Standard Roller Tolerances  ##

rOLLER_TYPE$R   tolerance(linear, d$R)  tolerance(linear, l$R)

typeA                  0.05                   0.1
typeB                  0.10                   0.2
                                                          #2 ##
endSection: constraints
```

Figure 5–2: Tolerance Allocation for Standard Parts

As part of the prototype refinement process, the designer can assume that the roller is of known type (see Figure 5–2) and EDS will respond by inferring the roller's parameters and the tolerances on those parameters from the *catalogueTable* constraints within the MCDF.

Functional Tolerances and Prototype Adaptation

Chapter 3 described the process characterised by Gero as prototype adaptation [Gero *et al*, 1988; Gero & Rosenman, 1990]. For a designer using EDS, prototype adaptation involves the modification and combination of constraints represented by one or more MCDFs to create a constraint set which reflects the design space being explored. Thus, the required design parameters are not known to be represented *a priori* and cannot serve directly as referents for tolerances values to be allocated. To support tolerance allocation in prototype adaptation, a functional description, such as a required fit between two bodies, serves to relate a parameter value to its tolerance.

For example, consider a design problem recognised by the designer as requiring a solution in the form of a peg-in-hole assembly of two parts. The designer combines their own constraints with instances of constraints which are represented in the DKB such as that shown in Figure 5–3:

```
beginSection: constraints

catalogueTable  Standard Fits for 8mm Basic Size    ##

fIT_TYPE$F  tolerance(linear, iD%h$F) tolerance(linear, oD%s$R) minC$F

running                0.022                  0.015          0.013
slide                  0.015                  0.009          0.005
push                   0.015                  0.009          0.000
drive                  0.012                  0.006         -0.001

                                                                #1 ##

endSection: constraints
```

Figure 5–3: Tolerance Allocation for Functional Fit

The *catalogueTable* in Figure 5–3 is taken from a generic fit MCDF, with typical name F, which is related by aggregation to two MCDFs representing a

shaft and hole with typical name variables s%F and h%F respectively. As part of the prototype adaptation process, the problem space instances are equated to the generic shaft and hole instances and an assumption made about the functional fit type required for the specific solution. EDS will respond by inferring tolerances for the assumed parameter values and the required tolerance fit.

As a specific example, consider the principal functional assembly of a Lucas Automotive 'inline' fuel pump which is called a pumping element. The two parts of the pumping element are called the plunger and body. To use the *catalogueTable* in Figure 5–3 to infer the tolerances on the external diameter of the plunger and internal diameter of the body requires assumptions relating parameters of the plunger to s%F and parameters of the body to h%F. In addition the designer would assume the type of fit (fIT_TYPE$F) required: for the pumping element this would be 'slide'.

Note that the discrimination between *catalogueTable* constraints for a given basic size parameter, the derivation of the maximum clearance parameter for the fit, and the choice of the size basis for the fit (whether shaft or hole-based) have not been addressed here. In practice the complete representation of standards for limits and fits (such as the British Standards Institution's BS4500 [BSI, 1984]) within the DKB is possible with the existing EDS constraint language. In other words, given a basic size and functional fit type, EDS is able to use existing standards to infer, and so define, the fit in terms of the parameters shown in Figure 5–1.

In addition to the use of standards, tolerance allocation is based on an individual designer's experience. Typically, a designer's assumption may be directly represented within a design support system, but the experiential justification which underpins it will not[1]. DSSs by adopting an AI-based approach can provide tools

---

[1]This is a problem that has to be directly addressed by researchers in the field of design automation. Their systems need to be capable of both representing and applying design experience.

to support human designers as they *apply* their experience. For example, tolerance allocation during prototype refinement can be supported by including sets of successful parameter assignments within an MCDF for future use or through mechanisms for reviewing design specifications developed from an MCDF. Thus, EDS provides a framework within which previous design explorations and their results can be used to inform future work.

Another important aspect of the application of experience within the tolerance allocation activity is the examination of the assumptions and consequences of tolerance choices for historical design descriptions. To support this activity the consistency maintenance element of DSSs is essential. Geometry-based CAD systems cannot represent the distinction between a designer's assumptions and their consequences. This distinction is critical to the analysis of historical designs. The designer should not adopt the assumptions of a requirement which is irrelevant to the current design exploration nor neglect to examine consequences of those assumptions which are to be adopted. For one of the examples from Section 5.1.1, the consequent manufacturing difficulties associated with a set of tolerance allocation assumptions can be reviewed prior to the allocation of tolerances for any new design aimed at the same manufacturing facility.

5.2.2 Quality Loss Function

In Section 5.1.2 the use of the quality loss function in a number of ways to support the tolerance allocation activity was introduced. This section investigates the implementation of this support within the Design Support System (DSS) framework of the Edinburgh Designer System (EDS).

To admit quality loss function reasoning within EDS the constraint language has to be extended. In the same way that the tolerance constraint form associates tolerance values to parameters of design instances (see Chapter 3), so quality loss function properties must be associated with these parameters. The proposed quality loss function constraint is of the form:

$$\mathrm{qlf}(qlf\_property, parameter\_expr) = qlf\_expr$$

The *qlf_property* is limited to the reference or inferred values of deviation of a parameter from its nominal value or the loss associated with that deviation. The right hand side, the *qlf_expr*, may be either a value or an expression into which a value may be propagated (such as a tolerance expression). Given this extension to the constraint language, the definition of the loss function given as Equation 5.1 can be re-expressed for a parameter (p) of design instance (i) within EDS as:

```
qlf(loss, p$i) = (qlf(ref_loss, p$i) / qlf(ref_dev, p$i))
                 * (qlf(dev, p$i)**2)
```

To assess the usefulness of this proposed quality loss function representation, consider the three tolerance allocation tasks described in Section 5.1.2.

Tolerancing a Functional Requirement

EDS can use the proposed quality loss constraint form and the loss function expression to infer additional bounds on a functional requirement. Consider a tolerance statement of the form:

```
tolerance(linear, d$hole) = 0.018
```

(This tolerance value might have been allocated using the methods described in Section 5.2.1.)

Assume that should instances of the designed artefact deviate from the nominal beyond the bounds defined by this tolerance value then the artefact is rendered unusable. For example, a plunger seizes inside a bore. Thus, the tolerance statement defines a reference deviation (half the tolerance) with which a loss is known or can be estimated. In terms of EDS assumptions:

```
EDS> assumeEds(qlf(ref_dev, d$hole) =
                              0.5*tolerance(linear, d$hole)).
EDS> assumeEds(qlf(ref_loss, d$hole) = 40).
```

Now assume that any artefacts found to be defective could be replaced during manufacture at a known cost:

```
EDS> assumeEds(qlf(loss, d$hole) = 5).
```

EDS can infer from the three assumptions, in conjunction with the constraint form of Equation 5.1, a value for the deviation from the nominal which represents the economic breakpoint for replacing defective artefacts during manufacture. For this example:

```
qlf(dev, d$hole) = 0.003
```

This value represents an additional constraint on the parameter d$hole which should be satisfied during manufacture. It should not be interpreted as giving rise to a second tolerance value in conflict with the first.

Tolerance Design

The quality loss function can be used to derive a tolerance value directly through a reformation of the process of finding additional constraints for functional requirements.

Consider the choice of tolerance for the thickness of a structural member to be made of a new plastic. The loss function can be used in conjunction with material data to determine this choice. Reference values for the loss function calculation might be derived from the deterioration of strength with increasing temperature. Thus in terms of EDS assumptions we might have:

```
EDS> assumeEds(qlf(ref_dev, p$i) = 0.7).
EDS> assumeEds(qlf(ref_loss, p$i) = 20).
```

(For some parameter, p, of a design instance i.)

Now assuming that the loss associated with the failure of the designed artefact produced from the new plastic with these characteristics is only a fraction of the reference figure in the material specification:

```
EDS> assumeEds(qlf(loss, p$i) = 8.5).
```

EDS can infer from these assumptions, in conjunction with the constraint form of Equation 5.1, a value for the deviation from the nominal associated with this

estimated loss and this deviation can be used to constrain the tolerance on the parameter. Thus given:

```
EDS> assumeEds(tolerance(linear, p$i) = 2*qlf(dev, p$i)).
```

EDS will infer:

```
tolerance(linear, p$i) = 0.91
```

This value represents an allocated tolerance for p$i which forms part of the complete design description for the artefact: should a different value for the tolerance be assumed or inferred, EDS should recognise this as a *valueConflict* inconsistency.

Inspection and the Loss Function

EDS can substitute assumed values for the process standard deviation and loss associated with a reference deviation of a parameter from its nominal value to determine the expected loss. To determine the loss associated with complex cases where inspection or reworking affect the process standard deviation and loss, additional constraints and reasoning capabilities are required. For example, we might constrain the loss with inspection to be equal to the sum of the inspection cost, the cost of defective fraction identified by the inspection, and the loss associated with the acceptable deviations from nominal of the remainder.

The quality loss constraint form proposed in this section is inadequate to support this type of reasoning. Complex quality loss analysis requires the definition by the designer (or DKB-builder) of additional quality loss properties. In effect, the requirement is for a property-of-parameter equivalent of the ability to declare addition design instance parameters which EDS currently supports.

Rather than complicate the proposed quality loss function representation, consider one 'work-around' which can be implemented within the existing EDS constraint language. For many engineering design problems where only one or two parameters of the MCDF instance are known to have a major impact on product quality, the declaration of additional parameters to represent quality loss proper-

ties provides a practical solution. Figure 5–4 presents a cut-down version of an MCDF to support this analysis. (An example of a full MCDF listing is given in Appendix A.)

Figure 5–5 demonstrates how a designer's assumptions together with instances of constraints #1 ## to #4 ## from the eXAMPLE MCDF are used within the EDS Design Description Document (DDD) to infer the quality losses associated with the variations in inspection proposed. In this example, taken from [Taguchi *et al*, 1989], the loss associated with inspection is greater than that for the case where no inspection is specified and no inspection cost thereby incurred. Clearly, the assumption-based maintenance of the evolving design description's consistency supports the designer's exploration of the relationships between the tolerance values allocated — the reference acceptable deviation from nominal — and the quality loss consequences.

5.3 Discussion

In Chapter 3 a justification for the study of the tolerance representation and reasoning task was presented. Tolerance statements, it was claimed, are significant because they relate the designed artefact to the design description. In particular, tolerance statements serve as a basis for testing designed artefact instances against their design description. To what extent are the tolerance allocation methods presented in this chapter adequate to support this activity? Several observations can be made:

- *Tolerancing functional requirements is not easy.*
  Section 5.2.1 demonstrated how DSSs such as EDS could support the allocation of tolerances to satisfy functional requirements. Unfortunately, many requirements do not yield to the kind for solution proposed for allocating tolerances to fit types. For example, tolerances required to satisfy specific aesthetic requirements or styling considerations are not readily formalised.

```
    class name [X]: eXAMPLE

beginParameters

    lossd$X          real    -       #1 Loss associated d ##
    lossdi$X         real    -       #2 Loss associated inspect d ##
    ref_devd$X       real    mm      #3 Reference deviation ##
    ref_lossd$X      real    -       #4 Loss at reference deviation ##
    sdd$X            real    mm      #5 Process sd for d ##
    sddi$X           real    mm      #6 Inspected process sd for d ##
    lossi$X          real    -       #7 Loss associated with inspect ##
    ndi$X            real    -       #8 Fraction rejected on inspect ##

endParameters

beginSection: constraints

    lossd$X =  (ref_lossd$X/(ref_devd$X**2))
              *(sdd$X**2)                                  #1 ##

    lossdi$x =   lossi$X + (ref_lossd$X*ndi$X)
               + ((ref_lossd$X/(ref_devd$X**2))
                 *(sddi$X**2))                             #2 ##
/*
    Statistical constraints are of the form:
        ndi$X  = f1(ref_devd$X, sdd$X)                     #3 ##
        sddi$X = f2(ref_devd$X, sdd$X)                     #4 ##
    Not included in this cut-down version
*/
endSection: constraints
```

Figure 5–4: Inspection and the Loss Function MCDF

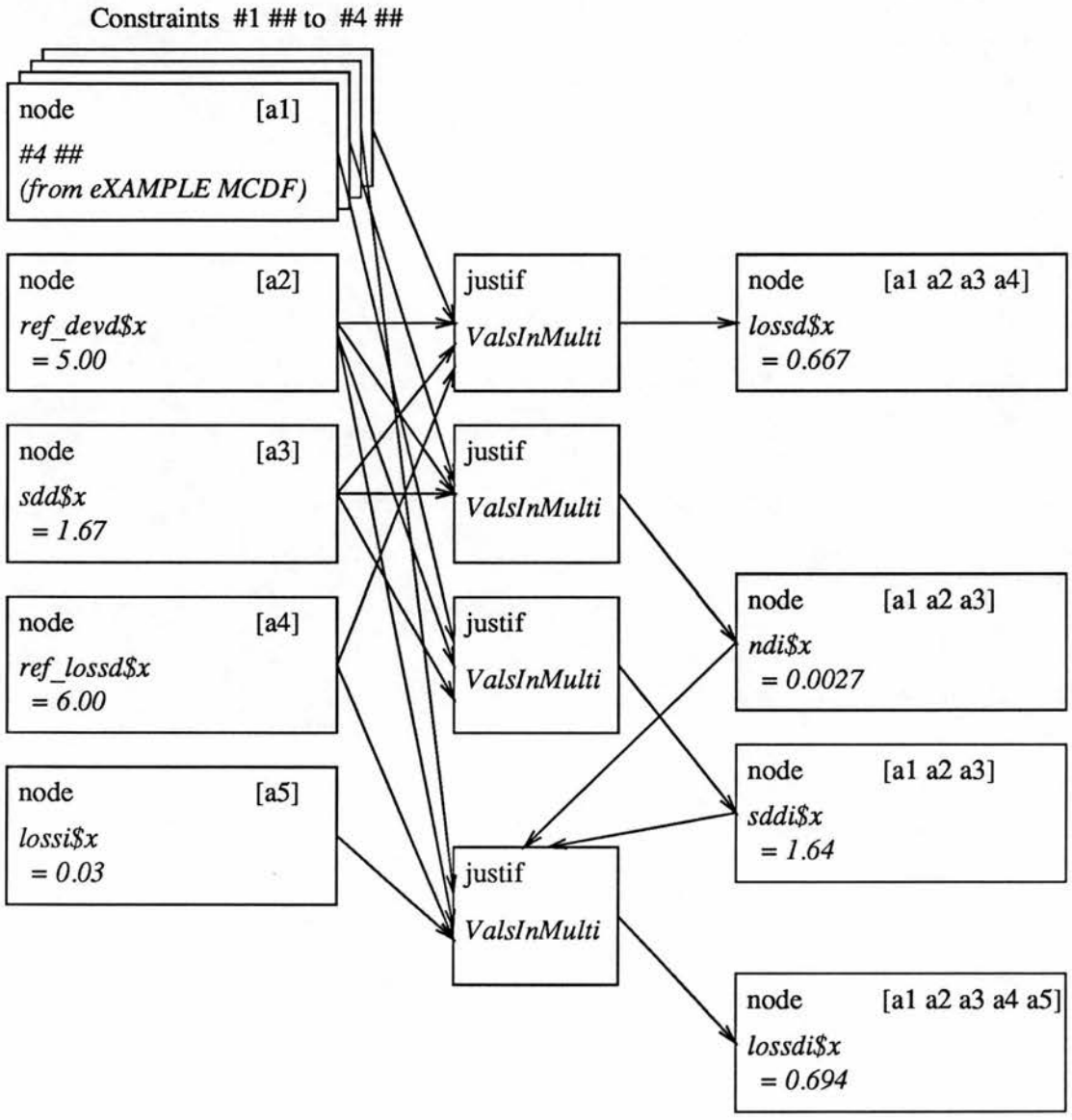


Figure 5–5: Inspection and the Loss Function DDD

- *The definition of a tolerance statement must be clear.*

  Section 5.2.2 illustrated the use of the loss function to allocate tolerances, but also drew attention to the way in which several measures of deviation from a nominal value are significant. For example, in the proposed EDS notation: *dev* and *ref_dev* as well as the linear tolerance. In Taguchi's original loss function description [Taguchi, 1986], there is considerable equivocation as to which figure is the tolerance. In the work presented here, the tolerance value forms part of a design description developed from an initial requirement statement. A tolerance statement does not represent a quality engineering constraint which if violated produces functionally-satisfactory designed artefacts uneconomically.

- *Finding functional constraints is hard.*

  Assuming or inferring tolerance constraints for the principal performance-determining parameters is one of the major tasks for the designer. This chapter has demonstrated how DSSs can support the designer in this task. How the many other parameters of the design description (which have less significant effects on the performance stipulated by the requirements statement) are allocated tolerance values has not been addressed. Without tolerance statements for all design parameters, designed artefact instances cannot be tested against their design descriptions. There are two solutions: either devise methods for complete tolerance allocation or revise the definition of an acceptable designed artefact. Current design practice adopts the first solution in the form of default linear and angular tolerances which apply to all parameters "unless otherwise stated". Chapter 7 includes a proposal for investigating the second alternative based on the elements of AI-based design.

In summary, this chapter introduced in Section 5.1 two approaches to the tolerance allocation activity: the use of standards and experience, and the application of the quality loss function. The provision of support for these approaches within DSSs was described in Section 5.2 and illustrated with examples from, and pro-

posals for, the use of EDS. The chapter closed with a discussion of the limitations of these tolerance allocation methods.

The next chapter presents design support as an activity which is necessarily situated within a product creation process. This realisation is essential if the activities of tolerance combination and allocation, the subjects of Chapters 4 and 5, are to be effectively supported by DSSs such as EDS.

Chapter 6

Situated Design Support

This chapter examines the improved provision of support for human designers within the wider context of the product creation process. The chapter has three sections. In the first section, a number of approaches to integrating the application of computer-based systems with the product creation process are outlined. The role of design descriptions within product creation is the subject of the second section. The use of design descriptions to produce process and assembly plans is also discussed. In the third section, a practical demonstration is presented of how a design description which is not geometry-based can be used as the basis for deriving consequences of use outside the design environment. As a specific example of this, the generation of parts list information from the Design Description Document of the Edinburgh Designer System is demonstrated.

6.1 Approaches to Integration

Computer-aided design (CAD) involves the use of a computer to assist in certain types of design activity; for example, developing, analysing, or modifying an engineering design [Groover & Zimmers, 1984]. CAD systems are characterised by being geometry-based. Indeed, many CAD systems can be viewed as aids to draughting only. Despite this limitation, CAD systems are now common features in industrial design offices principally because of the improvements they offer in designer productivity and drawing standardisation.

Computers have also been used to assist in the manufacturing process (CAM) and to automate or support process planning[1]. Attempts have been made to bring together the various computer-based systems to assist in the product creation process; often under the labels of CAD/CAM or CIM (computer-integrated manufacture).

The motivation for CIM is the desire to reduce or remove the *knowledge loss* which occurs at each activity boundary in the sequential data flow model of product creation. For example, the complete product knowledge generated by the design engineering activity is not passed to the process planning activity. This knowledge loss process is shown in Figure 6–1 which is taken from [Smithers, 1985]. Computer-based tools have been seen as the solution to this problem.

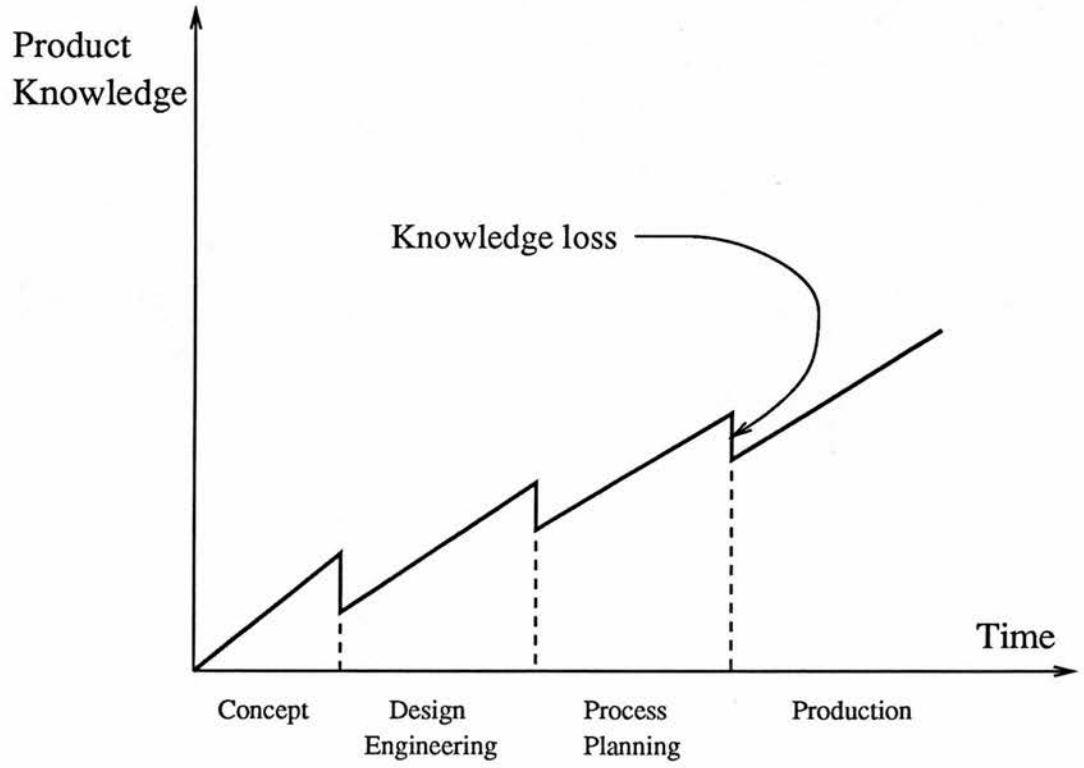


Figure 6–1: Knowledge Loss during Product Creation (from [Smithers, 1985])

[1]A brief explanation of computer-aided process planning (CAPP) is included in the survey of related work presented in Chapter 2.

Although all the required acronyms have been generated, the integration of computer-based systems to support the product creation process has not been wholly successful. The principal reason for this is that the development of systems for the various activities within product creation has proceeded without the consideration of integration requirements. The result has been to expose the frailty of the data flow model as a basis for building computer-integrated systems for product creation. Specifically, system capabilities and robustness are inherently limited by the data transferred from one sub-system to the next. This type of CIM system is presented in Figure 6–2 which shows how computer-based tools support the product cycle model presented in Chapter 1.

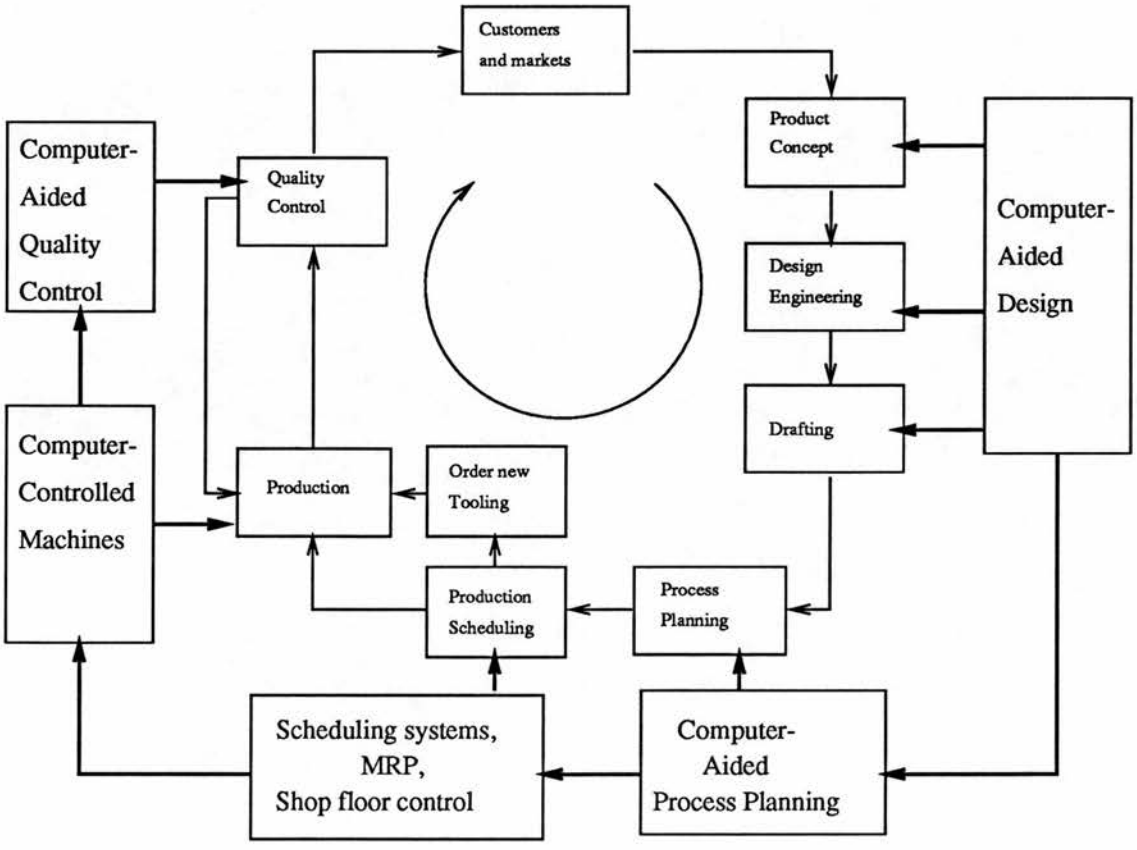


Figure 6–2: Computer-based Support for the Data Flow Model

Despite the limitations of the data flow model, its apparent decomposition of product creation into disjoint activities and its simplified treatment of the temporal relationships between these activities is appealing. For these reasons, the CAD/CAM model remains the background for much current research in computer-

integrated manufacture. For example, Ito *et al*'s work [Ito *et al*, 1988] proposes the use of expert systems technology to produce manufacturing data associated with a geometric drawing in advance of subsequent process planning activity.

An improvement on the data flow model makes use of a Product Knowledge-Base. This knowledge-base contains both the relatively static background to the design domain and the dynamic product description of the current design. As a result, product knowledge-bases are characterised by their data intensity and data complexity [Voelcker *et al*, 1988]. Static knowledge includes, for example, legislative regulations and design histories which must be available to all activities within product creation. Indeed, the list of perspectives which the product knowledge-base must support is long. For example, the dynamic product data should include financial and commercial consequences of design activity decisions [Carter, 1990]. The product description is built-up as activities are completed. For example, design engineering and process planning activities would contribute to the product description prior to most meaningful activities in the production environment (see Figure 6–3).

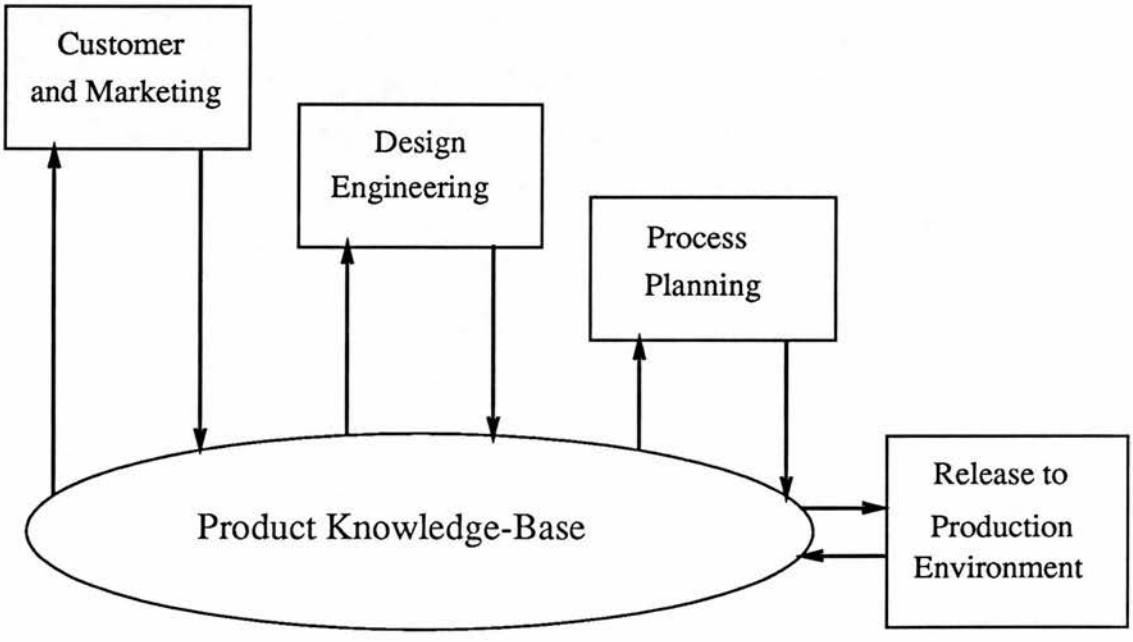


Figure 6–3: Product Knowledge-Base within the Design Environment

Note that including the Product Knowledge-Base in the model of product creation generalises the notion of a product creation activity from a distinct activity

in a fixed sequence of disjoint activities in the data flow model to any activity which is capable in some way of adding to or amending the product description.

In addition to developing the definition of a product creation activity, the introduction of the Product Knowledge-Base raises a number of practical difficulties for controlling product creation. The sequential framework of the data flow model has now been replaced and the explicit interfaces between particular activities which have historically served as control and review points within product creation have been lost. For example, Lucas Automotive require a failure modes and effects analysis to be completed before a design is released to production.

Responding to these problems, the *Design-to-Product* project [Smithers, 1985] introduced a Product Knowledge-Base whilst retaining an essentially data flow framework (see Figure 6–3). This system design decision recognises that in order to realise the benefits of a Product Knowledge-Base, solutions to the control problems outlined need to be found and implemented. Despite this pragmatism, the adoption and implementation of Product Knowledge-Bases within existing product creation environments is hard [Robertson, 1990].

One candidate control framework for product creation is to extend the blackboard architecture of the Edinburgh Designer System (EDS) from the design exploration process to the product creation process within which it is situated[2]. Under such a scheme, the central blackboard data structure would be the dynamic product description and the Knowledge Sources (KSs) contributing to the product description's development would be the various product creation activities. The number and competence of these KSs would define the coverage of an individual blackboard system's support. This coverage potentially includes activities from the design, production, and business environments (Figure 6–4). In addition, there is no requirement for the coverage of each environment or the

[2]See Chapter 3 for a brief presentation of the EDS architecture and [Hayes-Roth, 1985] for blackboard systems generally.

activities within them to be disjoint as KSs can co-operate or compete to make contributions to the product description.

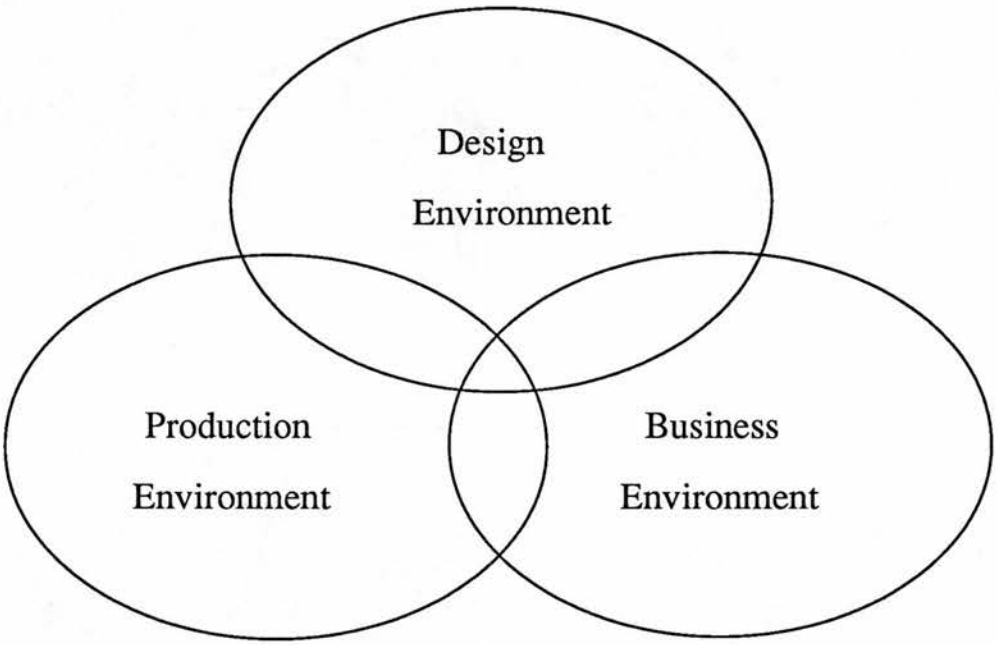


Figure 6–4: Three Environments for Product Creation Activities

The task of usefully scheduling and focusing the contributions of co-operating or competing KSs is that of providing *intelligent control* and is problematical. A blackboard architecture does provide a framework within which the contributions of human users and computer-based tools can be effectively integrated.

An alternative to the blackboard architecture for product creation is implied by the work of Winograd and Flores on the design of computer systems [Winograd & Flores, 1986]. For Winograd and Flores, product creation is a network of conversations between humans about actions *"such as request/promise, offer/acceptance, and report/acknowledgement"*. Hence to provide computer-based tools to support product creation involves not only the provision of systems for the use of individuals within an organisation, but also the provision of tools to support collective work in terms of the changing network of linguistic acts which constitute the product creation process in action. Intelligent control within this framework becomes the task of monitoring the network of commitments between humans within an organisation.

6.2 Using Design Descriptions

In the previous section, several approaches to the integration of tools for supporting activities within product creation were outlined. One of the major distinctions was the treatment of the design description. In the data flow model of Figure 6–2 the design description is passed to the computer-aided process planner whilst in Figure 6–3 the design description resides in the Product Knowledge-Base and is thereby made available to other tools. Underlying these differing treatments of the design description are different expectations of how the design description is used. This section discusses the use of design descriptions in product creation, specifically their use in planning activities in the production environment.

6.2.1 Process Planning

In the data flow model of product creation, process planning involves deriving a plan for manufacture from a geometry-based design description, typically an engineering drawing. The design description is thus used directly by the computer-aided process planning (CAPP) tool or the human expert.

Developing improved CAPP tools will require extended design descriptions which are not geometry-based. For example, if manufacturing features are to be used in the derivation of a process plan, these features need to be related to the design description. In addition, dependency structures within a design description can inform the planner by discriminating between parameter and tolerance values of the design description chosen for supposed ease of manufacture and those derived directly from a functional requirement [Requicha & Vandenbrande, 1988]. Thus, the successful integration of the process planning task within product creation is constrained by both the automated reasoning capability of the planning system and the knowledge representation of the design description.

6.2.2 Assembly Planning

Assembly planning involves the derivation of the plan for assembling a part from a collection of component parts. This process is complex, as products such as automotive vehicles are assembled from thousands of components, many of these components being themselves assemblies. In addition, assembly process characterisations are less well understood than those for part-manufacturing such as machining [Voelcker *et al*, 1988].

Design descriptions include statements about the shape of the component parts of an assembly, tolerances on the form and location of component part features, and the spatial relationships between the parts in the completed assembly. In other words, the design description focuses on the required functional assembly not how this required functional assembly is realised. For example, the principal part motion consideration in the design of a shaft and bearing is rotational whilst the assembly of the shaft within the bearing is achieved by an axial motion.

One consequence of representing the functional assembly of parts within the design description is that the aggregation hierarchy adopted by the designers, or the design knowledge-base builders, may not be the hierarchy of assemblies and sub-assemblies developed by the assembly planner. In other words, although the design description contains part structuring constraints this aspect of the design description cannot be used directly by an assembly planning tool[3]. The derivation of parts lists from design descriptions which are not geometry-based and which include part-structuring information is also subject to the constraint that, in general, the collections of components parts do not correspond to assemblies. (See Section 6.3 for a discussion of this problem.)

---

[3]Design knowledge-base builders can circumvent this problem for *existing* products by reproducing the known assembly structure through aggregation in the design knowledge-base. In terms of the kinds of design described in Chapter 3, knowledge bases of this type can support prototype refinement only.

6.2.3 Continuous Improvement

In Figure 6–2, the underlying data flow model includes a single feedback path from quality control to production. This path represents, for example, how the inspection of parts can be used to determine whether existing manufacturing processes are appropriate and so lead, if necessary, to changes in the production environment. Arguably, Figure 6–2 could be improved by adding more feedback paths. For example, recognising that a design review frequently follows initial process planning prior to production scheduling.

A more general approach makes use of the Product Knowledge-Base of Figure 6–3 and dispenses with pre-specified feedback paths by adopting instead the concept of *continuous improvement.* Continuous improvement involves re-addressing activities in product creation throughout the lifetime of the product. Using the design description is the key to supporting this process. One common feature of continuous improvement is the *engineering change request.* Often an engineering change request proposes a design modification, such as an alternative tolerance value, to facilitate an activity — typically outside the design environment — which will improve product value (either improve product quality or reduce cost). At Lucas Automotive a typical change request might suggest a tolerance value change to support an alternative manufacturing process proposed by a new component supplier.

Assuming that design change control mechanisms exist within the business environment, acceptance of a change request follows an investigation of the relationship between the proposed change, the existing design specification, and the requirements statement which the design specification satisfies. Thus, successful application of the continuous improvement approach will depend upon access to this relationship between requirements, specification, and proposed change. For geometry-based design descriptions, the dependency structure which constitutes this relationship is 'knowledge lost' during product creation (recall Figure 6–1) and has to be re-created by product designers from the original requirement using their expertise. For AI-based design descriptions, this dependency structure,

including the designer's original assumptions, may be explicitly represented and can be used to support the change request investigation.

6.3 An Example: EDS and Parts List Generation

The previous section described in general terms how knowledge represented within a design description can be used. This section examines in detail one such use: the derivation of lists of parts from a design description.

Lists of parts (often referred to as LOPs) are one of the most important documents derived from the design environment and subsequently referred to in the production and business environments. For example, parts lists are used to control the purchase of raw materials and components, and to regulate the modification of products as components are altered.

The precise form of the parts list can vary, but the following features are fairly standard:

- The parts list is particular to a version of a specific product type. Thus, a parts list records the versions of the constituent parts and has its own version number.

- For each component in the parts list there is a corresponding part number, textual description and the quantity value. The quantity value represents the number of instances of that component in the design description.

- In addition to an inventory of parts, the parts list can, optionally, carry information about the assembly structure of the complete product. For example, a *level number* would give an indication of the position (depth) of the component within the parts list tree structure. A *line sequence number* is an indication of the ordering in which assembly operations are performed. Hence if the parts list is ordered by line sequence number, multiple entries

are required for a single component number if the part appears in different sub-assemblies.

Within EDS there are limits on the type of parts list which can be generated from the information available within the Design Description Document. These constraints naturally affect the specification of the parts list tool for EDS and how the tool can be used outside the design environment.

6.3.1 Parts List Tool for EDS

The basic requirement for the Parts List Tool is to build the list of parts (LOP) for a particular instance label in the Design Description Document (DDD). Two variations of this top-level Prolog predicate are provided: the first `buildLOP/3` builds the parts list for a specific label whilst the second, `buildLOP/2` finds all the instance labels in the design description and builds a parts list for the list of these labels[4]:

```
buildLOP(InstanceLabel, ContextTagList, File) :-
            welcomeToLOPtool,
            tagListToContext(ContextTagList, Context),
            executeableLOP(InstanceLabel, Context, InstanceList),
            buildPartsTree(InstanceList, Context, LOPpartTree), !,
            writePartsToFile(LOPpartTree, File),
            quitLOP(File).


buildLOP(ContextTagList, File) :-
            welcomeToLOPtool,
            tagListToContext(ContextTagList, Context),
```

[4]The /n notation is used by Prolog programmers to indicate the number of arguments or arity of the predicate.

```
collectAllInstances(Context, AllInstances),
buildPartsTree(AllInstances, Context, LOPpartTree), !,
writePartsToFile(LOPpartTree, File),
quitLOP(File).
```

The arguments of the predicate `buildLOP` are: the instance label for which the parts list is to be built (`InstanceLabel`), a specification of a portion of the Design Description Document to which the Parts List Tool should confine its attention (`ContextTagList`) and the name of the file into which the resulting parts list will be written (`File`).

EDS is implemented in a mixture of POP-11 and Prolog programming languages using the Poplog multi-language programming environment [O'Shea & Eisenstadt, 1984]. Compatibility with EDS and ease of accessing the DDD data structure therefore suggested POP-11 [Barrett *et al*, 1985] and Prolog [Clocksin & Mellish, 1984] as the candidate languages for implementing the Parts List Tool. The Parts List Tool is implemented in Prolog as this was considered more appropriate for the recursive construction of the parts list from the *part-of* relation tree structure[5].

The goal of creating the parts list involves 3 major sub-goals: retrieving the fragment of the design description from which the parts list is to be built, building the parts list data structure, and writing the results to a file in the required format. These three tasks are achieved by the predicates `tagListToContext`, `buildPartsTree`, and `writePartsToFile`, respectively. Of particular interest is the procedure for building the parts tree, `buildPartsTree/3`, which recursively tests the potential parts of a label building the labels which satisfy the defined

---

[5]Implementation of the EDS Parts List Tool for use within the *Design-to-Product* system was undertaken by Bing Liu at Edinburgh from the author's requirements description. The detailed description of the Parts List Tool presented here relates to a later version of the tool developed by the author.

partness criteria into the part tree under construction. In this case the partness criteria are implicit in the `testLabelForPartness` goal which confirms that the required parameter declarations and acceptable parameters values are present in the design description context.

```
buildPartsTree([], _, []).

buildPartsTree([InstanceLabel]:FileName, Context,
               [[[InstanceLabel,FileName,NumbShp],Declaration,Values],
               SubPartTree]) :-
           testLabelForPartness(InstanceLabel, Context,
                                NumbShp, Declaration, Values),
           getCandidateSubParts(InstanceLabel, Context, CandidateData),
           buildPartsTree(CandidateData, Context, SubPartTree).

buildPartsTree([InstanceLabel|InstanceList], Context,
               [HdPartTree|TlPartTree]) :-
           buildPartsTree(InstanceLabel, Context, HdPartTree),
           buildPartsTree(InstanceList, Context, TlPartTree).
```

Thus, the Parts List Tool effectively extracts a sub-tree of the *part-of* structuring relation for the desired instance label producing a text file reflecting the parts list of the design description context. For example, for a single component represented by an aggregation of design features, the parts list should contain an entry for the component, but none for the design features. Two examples of the use of the Parts List Tool are described in the next sub-section.

6.3.2 Using the Parts List Tool

The first example of the use of the Parts List Tool is straightforward. The general arrangement of the major components of a fuel injection pump of a certain type is shown in Figure 6–5.

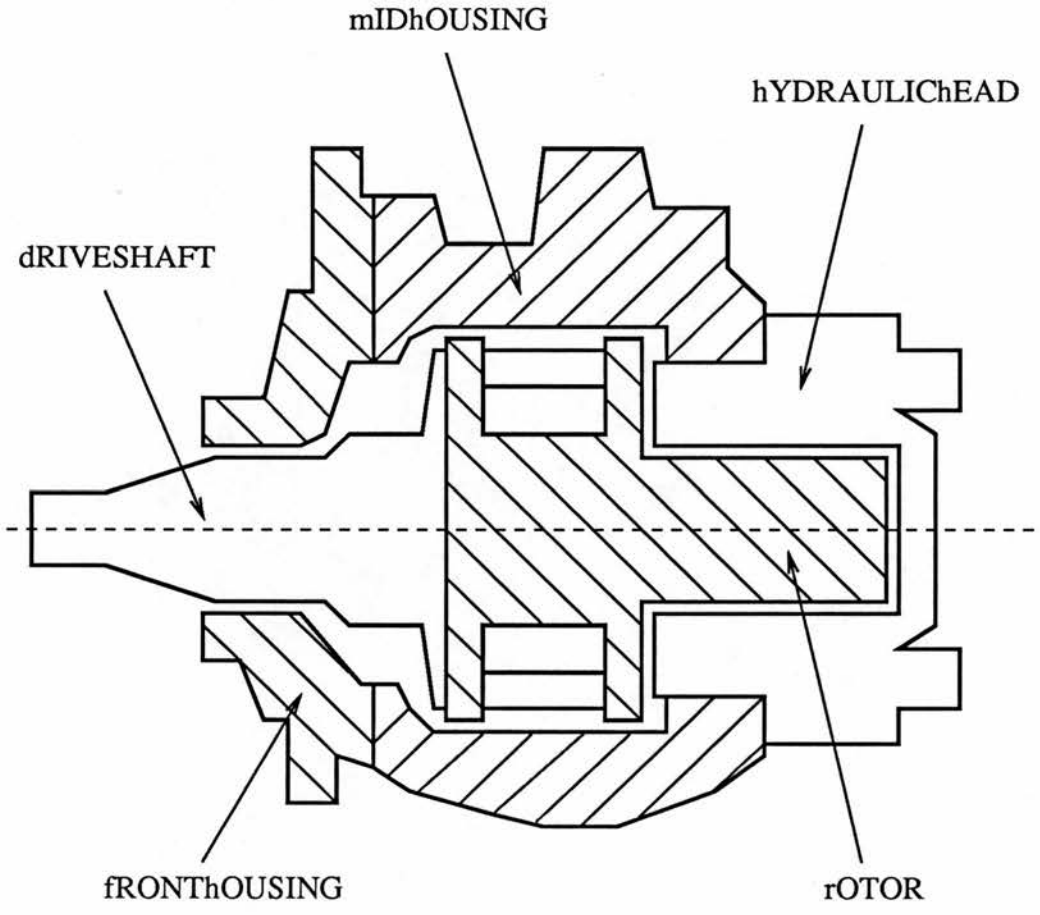


Figure 6–5: Sketch Derived from the pUMP MCDF Shape Constraint

Within the DKB this arrangement is represented by a top-level Module Class Definition File (MCDF) called pUMP. This MCDF is related by the *part-of* structuring to mIDhOUSING, rOTOR, fRONThOUSING, dRIVESHAFT and hYDRAULIChEAD MCDFs as shown in Figure 6–6. These MCDFs represent the candidate parts of the pUMP MCDF and, in this simple case, all the candidates are intended to represent the physical parts of the pUMP and are therefore admitted to the parts list.

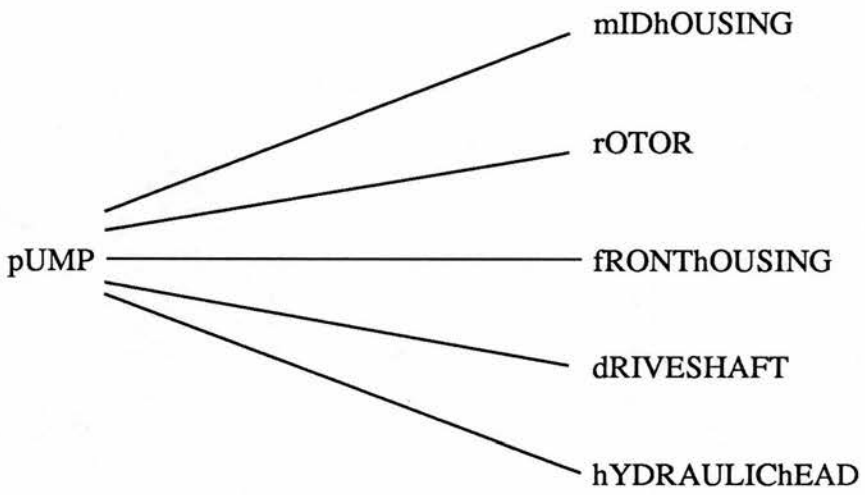


Figure 6–6: Aggregation Relationship for the pUMP MCDF

The parts list which results from applying the Parts List Tool to a design description based on the pUMP MCDF is shown in Figure 6–7. Note the retention of the EDS instance label as an identifier and the introduction of a Level Number based on the number of *part-of* structuring relations between the particular instance and the top-level instance of the parts list.

The second example of the use of the Parts List Tool is less straightforward and requires the discrimination between those MCDFs which represent parts and those which do not.

Line Number	Instance Label (from DDD)	Part Number	Level Number	Part Name (Name of EDS MCDF)
1	x	xp4984_6	0	pUMP
2	dshaft%x	xp261	1	dRIVESHAFT
3	front%x	xp3905	1	fRONThOUSING
4	head%x	xp2618B	1	hYDRAULIChEAD
5	mid%x	xp2468	1	mIDhOUSING
6	rotor%x	xp2646M	1	rOTOR

Figure 6–7: Parts List for the pUMP MCDF

One activity within the design of a type of fuel injection pump is the analysis of the pumping line. The pumping line is an area of interest within the pump design rather than a particular sub-assembly. For example, only the profile of cam rather than the entire cam need be considered for the pumping line analysis. The general arrangement of the components of the pumping line is shown in Figure 6–8.

Within the DKB this arrangement is represented by a top-level Module Class Definition File (MCDF) called pUMPINGlINE. This MCDF is related by the *part-of* structuring to pLUNGER, rOLLERaSSY and pROFILE MCDFs which represent the physical bodies within the pumping line. In addition, two MCDFs called cONTACT_CYLpLANE and cONTACT_CYLcYL are used to provide collections of constraints for solving the contact stress and endurance equations between physical bodies. Finally, the rOLLERaSSY is itself an aggregation of MCDFs, including another instance of the cONTACT_CYLcYL MCDF, as shown in Figure 6–9.

The fifteen MCDFs which aggregated together represent the pumping line are the candidate parts of the pUMPINGlINE MCDF. Unlike the first example involving the pUMP MCDF, not all of the candidates are intended to represent physical parts of the pUMPINGlINE and are not therefore admitted to the parts list.

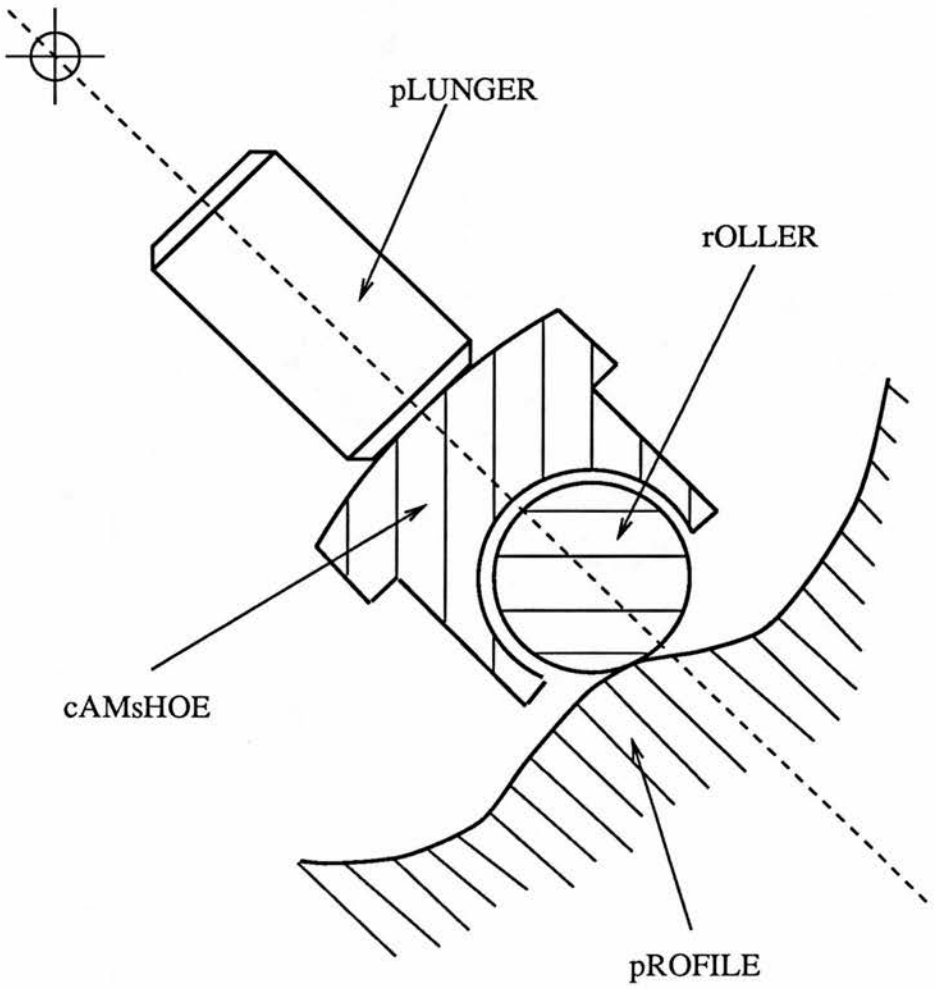


Figure 6–8: Sketch Derived from the pUMPINGlINE MCDF Shape Constraint

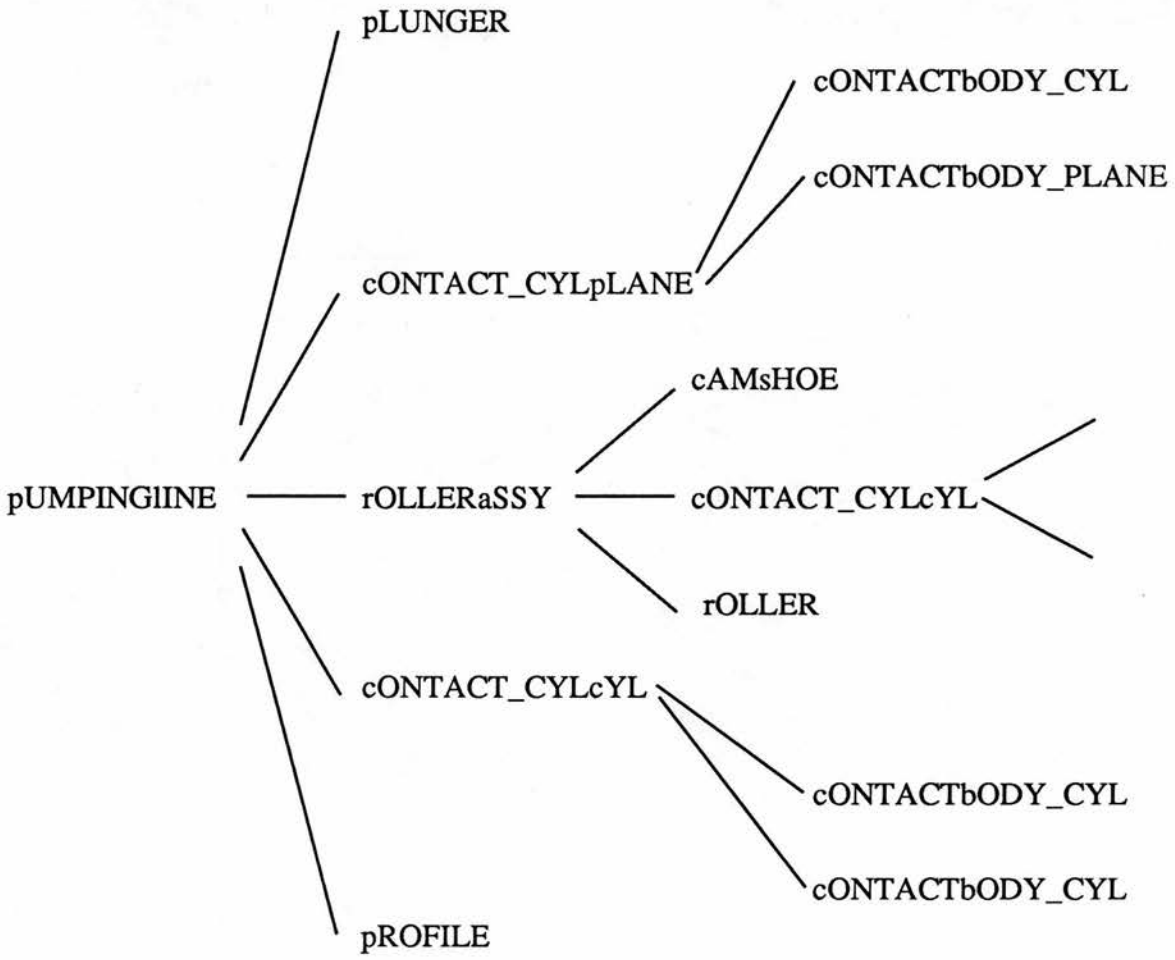


Figure 6–9: Aggregation Relationship for the pUMPINGlINE MCDF

The parts list which results from applying the Parts List Tool to a design description based on the pUMPINGlINE MCDF is shown in Figure 6–10. Note the retention of the EDS instance label as an identifier despite the top-level label (x) not itself being admitted to the parts list. Note also that there are two entries at the top-level within the parts list: pLUNGER and rOLLERaSSY.

Line Number	Instance Label (from DDD)	Part Number	Level Number	Part Name (Name of EDS MCDF)
1	plunger%x	xp3649_7	0	pLUNGER
2	rollerAssy%x	xp187071	0	rOLLERaSSY
3	roller%rollerAssy%x	xp1832_1	1	rOLLER
4	shoe%rollerAssy%x	xp1867_2	1	cAMsHOE

Figure 6–10: Parts List for the pUMPINGlINE MCDF

6.3.3 Discussion

A number of outstanding issues arise from the description of the EDS parts list tool and its use to derive parts lists for the two example cases:

- *Finding distinct properties to test for partness is difficult.* The EDS Parts List Tool requires some distinction between the properties of different EDS labels such that these labels can be said to represent different classes of object such as parts, features, and part assemblies. The question arises, do these distinctions exist? In particular, the Parts List Tool makes use of the declaration of parameters for the labels within the DDD. Whilst these declarations may derive directly from a designer's assumption or indirectly via an assumption of an instance of a MCDF from the DKB, there is no way of stating that a parameter is *not* a property of a label. In other words, it is impossible to distinguish between what is a 'non-part' and what is a 'part-

as-yet-uninstantiated' for any design description which is not understood to be complete.

- *Finding 'identical parts' and collecting them is hard.* In introducing the parts list, it was explained that one common feature is the quantity column where the number of bolts, screws or washers of a given part number (in that assembly) is recorded. Producing this information from EDS would be hard though desirable. What is required is a definition of sameness which can be applied to different labels, presumably by comparing parameter values which satisfy the partness criteria. This is difficult: for example, shapes would need to be compared without regard to location or orientation. As a result, a design decision was taken to retain the EDS label as a unique identifier within the DDD and the parts list.

- *Need to control the part definition process.* So far the discussion has suggested that any label which satisfied a simple conjunction of tests would be considered a part. Unfortunately, using this test alone certain undesirable consequences — acceptable to EDS! — could follow which would be unacceptable in the commercial environment. For example, multiple contexts allow one part number to be associated with more than one shape. Similarly, the same part number can be assigned to different labels in different design descriptions. The confusion arises from the need for unique identification of part descriptions across the design, production and commercial environments. Without control procedures to constrain the part definition process, neither the EDS label nor the part number can successfully fulfil the desired role.

- *There are severe restrictions on the valid use of EDS parts lists.* It is important to recall that the Parts List Tool in EDS tackles a subset of the issues addressed by the use of parts in the production and commercial environments. In particular, since the part-of structuring in the EDS parts list springs from the *part-of* relation used in the DKB, it is manifestly not the part-of structuring associated with assembly. Rather MCDFs, and hence

parts, are aggregated for functional purposes. Thus the level numbers in the EDS parts list do not reflect the line sequence numbers which could be inferred from an assembly plan. In addition, EDS does not provide the control tools required for the issue of new parts lists arising from changes to individual components, for example.

In conclusion, the Parts List Tool demonstrates that design descriptions which are not geometry-based can usefully serve as a basis for deriving consequences of use to activities outside the design environment. In addition, this work has highlighted the relationship between the parts list, the assembly plan, and the control issues associated with their derivation.

6.4 Summary

Supporting human designers successfully requires the recognition that engineering design activities are situated within the wider product creation environment; this *situated design support* formed the subject of this chapter.

In Section 6.1, different approaches to the integration of computer-based systems to support product creation were described. Section 6.2 discussed the use made of design descriptions by activities outside the design environment. Limitations in the representation of design descriptions, for example the absence of tolerance representation, were shown to constrain the useful integration of activities within product creation. The use of a design description which is not geometry-based, namely the Design Description Document of the Edinburgh Designer System, was examined in Section 6.3. In particular, the derivation of a parts list for use outside the design environment was demonstrated and discussed.

The next chapter concludes the thesis by reviewing the results of this and the two preceding chapters with reference to the objectives of Chapter 3 and the characterisations of AI-based design and Design Support Systems presented in Chapter 2.

Chapter 7

Conclusion

The previous three chapters have described a series of investigations into the support of tolerance combination and tolerance allocation activities and the situated design process. This chapter, which represents the conclusion of the thesis, summarises the outcomes of these investigations with reference to the experimental objectives presented in Chapter 3. In addition to a review of experimental results, this chapter contains a summary of the original contributions of the thesis, implementation details relevant to the work and some ideas for further work required in this research field.

7.1 Outcomes

This section presents the outcomes of the work presented in Chapters 4, 5 and 6 with reference to the experimental objectives derived in Chapter 3.

7.1.1 Using AI-based Design to Support Reasoning about Tolerance

Chapter 3 introduced the support of tolerance reasoning activities as a significant and valid area to compare the geometry-based and AI-based approaches to engineering design. In Chapters 4 and 5 methods for supporting tolerance combination and allocation were investigated. The use of design descriptions which

were not geometry-based together with automated reasoning facilities, such as those provided by the Relation Manipulation Engine in EDS, enabled tolerance combination through functional loop analysis and tolerance allocation through the use of standard components and functional tolerances to be demonstrated.

The adoption of design descriptions which are not geometry-based makes possible the representation of dependencies between design constraints and the designer's assumptions which underpin those constraints. The Design Description Document (DDD) in EDS provides this type of design description and also supports the representation of spatial occupancy through its constraint language. Indeed, many of the capabilities associated with geometry-based design support are provided by the Geometric Modelling Engine within EDS.

In addition to the specific advantages of using an Assumption-based Truth Maintenance System for exploring design alternatives and comparing the consequences of different assumptions, consistency maintenance provides a framework for recording design histories and controlling design description changes. Although computer-based tools exist to support these design history and change control facilities for the CAD/CAM data flow model, these tools are restricted by the geometry-based approach. The evidence of this thesis is that the AI-based approach will admit design description changes, such as tolerance value modifications, to be supported by identified designer assumptions and related to specific functional requirements.

To sum up, the four elements of AI-based design: automated reasoning, knowledge representation, intelligent control, and consistency maintenance have all been shown to contribute to supporting tolerance reasoning activities which are unsupported in geometry-based design. In addition, the experiments have shown that the spatial occupancy reasoning of geometry-based design can also be supported within the AI-based framework.

7.1.2 Using AI-based Design Descriptions Within Product Creation

One argument against the AI-based approach to design is the claim that improvements in design support are offset by difficulties introduced to the support of the product creation process. Chapter 6 tackles this criticism and demonstrates that EDS and its DDD, and by implication the AI-based approach, can be used to support activities outside the immediate design environment. In the case of List of Parts (LOP) generation, problems do arise in controlling the LOP generation process intelligently and discriminating between the kinds of design instance potentially present in a design description which is not geometry-based. Although, these problems do not occur in CAD/CAM systems, the impoverished nature of geometry-based representations rather than any limitation of the AI-based approach is the explanation.

7.1.3 Delivering AI-based Design Support through DSSs

Chapter 2 characterised Design Support Systems (DSSs) as: adopting an AI-based approach, providing support for human designers (in contrast to trying to automate design), using design process models, and having domain-independent architectures. Given that the AI-based approach improves support for human designers, can DSSs effectively deliver this improved support?

The DSS architecture adopted by EDS has been shown to provide support for the design of power transformers, direct current motors, spur gears and fuel injection equipment [Smithers *et al*, 1989]. In addition, the EDS architecture has been adopted for the design of manufacturing plans in a system called LUMP [Hinde *et al*, 1989; Herbert *et al*, 1990]. Note that manufacturing plan design requires alternative reasoning support systems as well as distinct domain knowledge. The utility of the EDS architecture in a number of domains is encouraging, as is the implied generality of the exploration-based model of the design process underpinning the EDS implementation.

The tolerance reasoning and LOP generation experiments with EDS were largely successful. As a consequence, the support of significant design activities within EDS has been demonstrated. Unfortunately, two aspects of these implementations have not been investigated: computability and usability. The two factors are related by their significant contributions to the overall effectiveness of the support system.

The number of distinct design domains in which EDS has been applied offers oblique evidence that the System Development Interface which EDS provides is usable. However, the EDS command language is cumbersome (see Appendix A) and has been unpopular with some design engineers. No attempt has been made in this thesis to investigate user interface issues or survey practising engineers to gauge the usefulness of the reasoning methods implemented.

Similarly, no attempt has been made in this thesis to investigate the computability of the methods to be implemented within EDS either theoretically, in terms of the algorithms used, or more practically, in terms of the scalability of the solutions for more complex design descriptions. The only evidence for the computability of the DSS solution is circumstantial, arising from the practical application of EDS to the support of fuel injection equipment design (see Chapter 3).

To sum up, although the experiments with EDS were successful, significant factors which have not been addressed by this thesis prevent the conclusion that DSSs like EDS can effectively deliver the improved support provided by the AI-based approach.

7.2 Original Contributions

The principal contributions provided by this thesis are:

Demonstration of AI-based Design's Improved Support

Following the characterisation of AI-based design in [Smithers, 1989], the motivating hypothesis that the AI-based approach improves support for human designers has been demonstrated. This was done through the experimental objective of showing how AI-based design can be used to support reasoning about tolerance (see Section 7.1.1). In addition, the generation of parts lists from design descriptions which are not geometry-based illustrates how the AI-based approach can be used to derive product creation information of use outside the design environment.

Framework for Tolerance Representation in AI-based Design

Tolerance statements are significant because they relate a designed artefact to its design description. The thesis explores this relationship using current industrial practice and related work to develop a framework for tolerance representation in AI-based design. Within this framework, tolerance reasoning activities are characterised by the modality of the inferences made as either tolerance combination or tolerance allocation. Tolerance combination includes reasoning about properties of the distribution of a designed artefact instances. Two approaches to tolerance allocation are presented: the use of standards and experience being contrasted with the systematic use of a quality loss function.

Experimental Results from Reasoning about Tolerance in EDS

In addition to providing a framework for tolerance representation in AI-based design, the thesis includes results arising from the experimental implementation of support for tolerance reasoning activities within EDS. Thus, for example, the

use of EDS to support tolerance combination using functional loop analysis and tolerance allocation using standards and experience is presented.

7.3 Current Implementation

The examination of the tolerance combination and tolerance allocation activities was undertaken between October 1985 and September 1988. The results of this work, which later formed the basis of Chapters 4 and 5, were influenced by three month placements in industrial design and manufacturing departments[1].

The construction of Declarative Knowledge Bases (DKBs) and the creation of Module Class Definition Files (MCDFs) to investigate the use of EDS to support design tasks dates from October 1988 to September 1990. This period includes both the MCDF implementation to demonstrate EDS *per se*, as illustrated in Chapter 3 and Appendix A, and the development of an EDS demonstration within the integration framework provided by the *Design-to-Product* project. (For details of the *Design-to-Product* system implementation, see [IEE, 1990].)

Throughout this thesis, the successes and limitations recorded in the use of EDS to support tolerance reasoning activities refer to experiments with, and the capabilities of, EDS at the time of the *Design-to-Product* project final demonstrations in March 1990. The then current version of EDS was known as EDSv7. (For details of the implementation of the EDS series itself see [Smithers *et al*, 1989].) Thus, proposed extensions to the EDS constraint language which were not implemented, for example the use of the Quality Loss Function for tolerance allocation, should be interpreted as providing enhanced support within EDSv7. The work with EDS reported in this thesis was undertaken at both the Department of Artificial Intelligence in Edinburgh and at Lucas Automotive Ltd. in Gillingham, Kent. EDS was mounted and running on a Sun-3 workstation with 8MBytes or 16MBytes of main memory.

---

[1]Arranged as part of the CASE Studentship supporting this research.

The discussion of the use of design descriptions in product creation which forms part of Chapter 6 derives from the author's experience as part of the team implementing the *Design-to-Product* final demonstrations. The specification and implementation of the Parts List Tool for EDS and the examples of its use were completed in September 1990. For compatibility with EDS, the Parts List Tool was implemented in the Prolog programming language [Clocksin & Mellish, 1984] within the Poplog multi-language programming environment [O'Shea & Eisenstadt, 1984]. Note that with the exception of the use made of EDS for tolerance reasoning, many of the implementational results of this thesis, such as the EDS DKB for fuel injection equipment design support, formed part of the archived results of the *Design-to-Product* project in 1990.

7.4 Further Work

The principal areas for further study suggested by the work of this thesis are:

Demonstration of Effective Delivery of AI-based Design Support

Following the characterisation of Design Support Systems (DSSs) in Chapter 2, the motivating hypothesis that DSSs constitute an effective means to deliver AI-based support to human designers was explored through the experimental use of EDS (see Section 7.1.3). A demonstration of the *effective* delivery of AI-based design support requires both an examination of the potential complexity of the reasoning methods required for adequate support and an investigation of how human designers can use these methods successfully. These comments suggest two areas of further work. Firstly, a theoretical study of the computability of tolerance reasoning methods including an evaluation of the interaction of these methods with consistency maintenance systems and control mechanisms within DSSs. Secondly, a system-based study of the use of DSSs by individual designers to identify realisable human-computer interactions for achieving tolerance reasoning tasks.

Improving Models of Product Creation

Although this thesis was motivated by a dissatisfaction with the data flow model of product creation, no attempt was made to develop an alternative. The results of this thesis provide several insights towards the specification of improved models of the product creation process. For example, the role of tolerance statements in product creation should be clear and this will provide one valid area to the test the claims of the proposed model against its CAD/CAM predecessor. The relationship between instances of a designed artefact and the design description which constrained its production must be clear. The model should support the use of tolerance statements to reason about processes outside the design environment as design descriptions evolve. Finally, the model should support reasoning about instances of designed artefacts during design so that properties of prototypes or manufactured populations of components can be referred to.

One radical basis for a model of the product creation process is to replace the concept of an artefact satisfying a design description with that of an artefact not conflicting with a set of product constraints. Thus, default tolerance values which cannot be justified by any element of an evolving design description would not be admitted. In the case where the simplification of manufacturing and inspection constituted support for the assumption of default tolerances, the explicit dependency of that tolerance on manufacturing rather than functional requirements would form part of the design description.

Extending Constraint Languages for Engineering Design

This thesis has not been directly concerned with specifying what might be called a constraint language for engineering design. Rather, the support of tolerance reasoning activities was attempted using the existing EDS constraint language. Where this language proved inadequate, such as in the representation of the Quality Loss Function, extensions to the constraint language were proposed. Thus, the EDS constraint language and the results of this thesis together provide valuable clues to the developers of improved constraint languages.

Note however that the results of this thesis are not sufficient. As the discussion of product creation models makes clear (above), the constraint language should support both reasoning about activities outside the design environment and the deliberate investigation of constraint conflict implied by an analysis of designed artefact failure modes and their effects. This thesis suggests that the development of improved constraint languages for engineering design should be driven by the results of using experimental DSSs and seen as an important part of designing better DSSs.

7.5 Summary

The work presented in this thesis was motivated by a dissatisfaction with the CAD/CAM data flow model of product creation and the geometry-based approach to design underlying it. Primarily through the study of tolerance representation and reasoning, this thesis has shown that support for human designers can be improved by moving from this geometry-based approach and toward an AI-based approach to engineering design. Future work should be focussed on the effective delivery of this improved design support and the development of alternative models of product creation which reflect the contribution of the AI-based approach.

Appendix A

EDS Details and Use

This appendix contains a complete listing of a Module Class Definition File (MCDF) which forms part of the Declarative Knowledge Base (DKB) used to demonstrate the solution of design problems with the Edinburgh Designer System (EDS). In addition to the MCDF listing, the appendix contains details of the user commands employed and system responses generated during the solution of a typical design problem (outlined in Chapter 3).

A.1 Module Class Definition File Syntax

The following is a complete and uninterrupted listing of the cAMrOLLERcONTACT MCDF. The MCDF is in three main sections: a header, a set of declarations, and a set of constraints. A brief explanation of the syntax of the MCDF follows the listing:

```
beginSection: moduleClassHeader

 beginClassName
    class name [Crc]: cAMrOLLERcONTACT
    created:    9/9/90
    created by: andy@GBP
 endClassName
```

```
beginModification
    modified:       24/9/90
    modified by:    andy@GBP
    reason:         Add comments for listing
endModification

beginIsAkindOf
    dESIGNfEATURE
endIsAkindOf

beginHasAsParts
    none
endHasAsParts


endSection: moduleClassHeader


beginSection: parametersVariablesConstants

beginParameters


/*
    These are the principle parameters of the cam roller contact:
*/

    rATED_SPEED$Crc         real    rad/s     #1 In rpm from engine ##
    lINE_PRESSURE$Crc       real    N/mm**2   #2 Outlet pressure ##
    b10_LIFE_REQ$Crc        real    s         #3 Requirement on cam ##

    nUM_OF_PLUNGERS$Crc     integer -         #4 Number of plungers ##
    pLUNGER_DIA$Crc         real    mm        #5 Diameter of plungers ##
    cAM_TYPE$Crc             -       -        #6 Cam type table access ##
```

```
    rOLLER_DIA$Crc          real    mm          #7 Diameter of roller ##
    rOLLER_LENGTH$Crc       real    mm          #8 Roller length ##
/*
    These are the principle geometric parameters of the cam:
*/
    cAM_LIFT$Crc            real    mm          #9  Available cam lift ##
    cAM_NOSE_RADIUS$Crc     real    mm          #10 Nose radius of cam ##
    cAM_RATE$Crc            real    mm/deg      #12 Rate of cam profile ##
    cAM_DEPTH$Crc           real    mm          #13 Diameter of roller ##
    bASE_RADIUS$Crc         real    mm          #14 Diameter of roller ##
    aLPHA$Crc               real    deg         #15 Attack angle on cam ##

/*
    Parameter to convert cam stress calculation units:
*/
    sTRESS_CONST$Crc     real    kg/(mm  s**2)    #16 Balance units ##

/*
    Additional parameters for b10 Life Derivation:
*/
    cAM_STRESS_TSI$Crc      real    N/mm**2     #17 In tsi ##
    b10_LIFE$Crc            real    s           #18 Cam life hours ##

    sAFETY_FACTOR$Crc       real       -        #19 Safety margin ##
    nUM_PLUNGERS_STD$Crc    integer -           #20 Number of plungers ##
    rATED_SPEED_STD$Crc     real    rad/s       #21 In rpm ##
    cAM_STRESS_STD$Crc      real    N/mm**2     #22 In tsi ##
    b10_LIFE_STD$Crc        real    s           #23 Cam life required ##

endParameters


beginVariables
    none
endVariables
```

```
beginConstants

    /*  Cam Types known to EDS  */
    typeA            -       -      #1 Used in current work ##
    typeB            -       -      #2 NOT used currently ##

endConstants


endSection: parametersVariablesConstants


beginSection: constraints

    catalogueTable  Cam Data Known to EdS (1 of 2)  ##

    cAM_TYPE$Crc cAM_RATE$Crc   cAM_NOSE_RADIUS$Crc aLPHA$Crc

    typeA           0.26              3               28
    typeB           0.16              5.5             19
                                                              #1 ##

    catalogueTable  Cam Data Known to EdS (2 of 2)  ##

    cAM_TYPE$Crc cAM_DEPTH$Crc  bASE_RADIUS$Crc     cAM_LIFT$Crc

    typeA           28            30                    1.65
    typeB           16            26                    1.5
                                                            #2 ##

/*
    Cam Stress and Life Calculation
    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
*/

cAM_STRESS_TSI$Crc **2  = sTRESS_CONST$Crc * (pLUNGER_DIA$Crc**2) *
                            (lINE_PRESSURE$Crc *
                          (cAM_NOSE_RADIUS$Crc + (rOLLER_DIA$Crc/2)))
                          /(cos(aLPHA$Crc*pi/180)*rOLLER_LENGTH$Crc
                          * cAM_NOSE_RADIUS$Crc * (rOLLER_DIA$Crc/2)
                          )                                          #1 ##

sTRESS_CONST$Crc = 12.062        #2 Introduce to balance units ##

/*
    Calculation of b10 life estimate is derived from empirical data
under a known set of conditions:

*/
    b10_LIFE_STD$Crc = 767           #1 In hours ##
    cAM_STRESS_STD$Crc = 135         #2 In tons per square inch ##
    nUM_PLUNGERS_STD$Crc = 4         #3 ##
    rATED_SPEED_STD$Crc = 1500       #4 In rpm ##

b10_LIFE$Crc =    b10_LIFE_STD$Crc
                * ((cAM_STRESS_STD$Crc / cAM_STRESS_TSI$Crc)**(20/3))
                * ( (rATED_SPEED_STD$Crc * nUM_PLUNGERS_STD$Crc)
                   /(rATED_SPEED$Crc * nUM_OF_PLUNGERS$Crc))    #2 ##

/*
    Finally, the derived value for b10 Life has to be better than
the requirement:

*/
b10_LIFE$Crc > sAFETY_FACTOR$Crc * b10_LIFE_REQ$Crc              #3 ##

/*
    The nominal shape of the cam roller contact is given by the
```

```
following shape expression:

*/

    shape$Crc  <==      (cyl(cAM_DEPTH$Crc,bASE_RADIUS$Crc)
                         /\ cyl(cAM_DEPTH$Crc,cAM_NOSE_RADIUS$Crc)
                              @ trans(bASE_RADIUS$Crc
                                      + (cAM_NOSE_RADIUS$Crc
                                           - cAM_LIFT$Crc),
                                      0,
                                      0))
                       \/  (cyl(rOLLER_LENGTH$Crc, rOLLER_DIA$Crc/2)
                              @ trans(bASE_RADIUS$Crc
                                        - (cAM_LIFT$Crc
                                            + rOLLER_DIA$Crc/2),
                                     0,
                                     0))                          #1 ##

endSection: constraints
```

The module class header contains the creation and modification history of the MCDF and specifies the location of the module within the kind-of and part-of structures of the DKB. In addition, the statement [Crc]: cAMrOLLERcONTACT defines the typical name variable, Crc, which is used to refer to the module in its constituent parts and declarations.

The second section declares the MCDF's parameters, variables and constants. Each declaration is of the form: name, type, dimensions, and number. Note that EDS uses an extended set of SI units (including millimetres for example) to define each parameter's dimensionality. As a result, the description of parameter units included within a declaration comment may appear to conflict with the parameter dimension declaration (the declaration of rATED_SPEED$Crc is an example). The remainder of the line, between the declaration number and the closing ##, is read as a comment and is used here to provide an explanation of each parameter

name. Every declaration is referred to the typical name variable Crc through the use of the $ symbol.

For the cAMrOLLERcONTACT MCDF, the constraints section includes examples of tables, real equations (including constraining parameter values), an inequality, and a geometric shape assignment. Note that the parameter shape$Crc used in the shape assignment constraint is not included in the declarations section of this MCDF, but is inherited as a consequence of the location of the module within the DKB.

A.2 Solving Design Problems With EDS

This section describes the use of EDS to solve the design problem introduced in Chapter 3. Two approaches to the problem solution are presented: prototype refinement and prototype adaptation.

In the design process represented here, the commands employed by the user follow the EDS> prompt. EDS-generated responses typically start with NEW > (exceptions to this are identified where they occur). Within a block of EDS output, a single line of 3 dots, ..., indicates that some lines are not shown in the interests of brevity. Within a single line of EDS output, the symbol ~ indicates that data has been compressed by EDS to fit onto that line. The EDS user interface allows such compressed lines to be expanded as required.

A.2.1 Prototype Refinement with EDS

Prototype refinement requires the existence of an MCDF which accurately reflects the design problem space. In this case, such an MCDF is found to exist and the solution begins with the assumption of an instance of this cAMrOLLERcONTACT module. This results in EDS creating an instance x of the cAMrOLLERcONTACT module in the Design Description Document (DDD) using the knowledge contained in the MCDF itself (described in the previous section). The creation of

the MCDF instance includes the inheritance of declarations from super classes, the creation of instances of the parameter declarations (not shown) and the creation of instances of the constraints relating those parameter instances:

```
EDS> assumeEds([x]: cAMrOLLERcONTACT).

NEW > N   1: [x] : cAMrOLLERcONTACT
NEW > J   0: assumption
NEW > K   0: findSuperClass (125)
NEW > K   1: findDirectParts (122)
NEW > K   2: findDirectConstraints (121)
NEW > K   3: declareLabels (124)
NEW > K   4: declareDirectAttributes (123)
NEW > N   2: [x] : dESIGNfEATURE
NEW > J   1: findSuperClass
RIP > K   0: findSuperClass (125)
NEW > K   5: findSuperClass (125)
NEW > K   6: findDirectParts (122)
NEW > K   7: findDirectConstraints (121)
NEW > K   8: declareLabels (124)
NEW > K   9: declareDirectAttributes (123)
NEW > N   3: [x] : hOMOGENEOUSoBJECT
NEW > J   2: findSuperClass
RIP > K   5: findSuperClass (125)
                    ...
NEW > J  14: findDirectConstraints
RIP > K   7: findDirectConstraints (121)
NEW > N  46: shape$x  <==  ~/\~ \/ ~@~
NEW > N  47: b10_LIFE$x  =  ~$x * ~**~ * ~**-1 * ~$x * ~**-1 * ~$x
NEW > N  48: b10_LIFE_STD$x = 767
NEW > N  49: cAM_STRESS_STD$x = 135
NEW > N  50: nUM_PLUNGERS_STD$x = 4
NEW > N  51: rATED_SPEED_STD$x = 1500
NEW > N  52: sTRESS_CONST$x = 12.062
NEW > N  53: ~**2 + -2*~*~*~*~*~*~ + -1*~*~*~*~*~*~*~  =  0
```

```
NEW > N  54: b10_LIFE$x  >  b10_LIFE_REQ$x * sAFETY_FACTOR$x
NEW > N  55: catalogueTable(row(~, ~, ~, ~),  [row(typeA, 28, 30, 1.65)
     row(typeB, 16, 26, 1.5)])
NEW > N  56: catalogueTable(row(~, ~, ~, ~),  [row(typeA, 0.26, 3, 28)
     row(typeB, 0.16, 5.5, 19)])
NEW > J  15: findDirectConstraints
RIP > K   2: findDirectConstraints (121)
NEW > K  15: bbTermination (50)
RIP > K  15: bbTermination (50)
aye
```

Within the EDS output, NEW > indicates that an item has been created within the DDD. The single letter which follows NEW > indicates whether the item is a datum node (N), a justification for a node (J), or a knowledge source activation record (K). The numbers in parentheses at the end of lines containing knowledge source activation records are ratings used to order the agenda used to schedule the work done by EDS. Where NEW > is replaced by RIP > this indicates that the work associated with an activation record has been completed. The creation of the MCDF instance is complete when the EDS blackboard terminates.

Following the assumption of the module instance, the designer assumes the parameter values provided by the initial requirement. In this design domain the initial requirement is directly related to customer's engine specification. Thus, values for the line pressure, rated speed and life requirement for the customer's engine are assumed:

```
EDS> assumeEds(lINE_PRESSURE$x = 720 and rATED_SPEED$x = 1500
   and b10_LIFE_REQ$x = 2000).
```

After assuming a value for the safety factor to be used for the design (not shown), the designer assumes that an existing cam specification can be used for this new application:

```
EDS> assumeEds(cAM_TYPE$x = typeA).
```

```
NEW > N  62: cAM_TYPE$x = typeA
NEW > J  19: assumption
NEW > K  19: valueInCatalogueTable (133)
NEW > K  20: valueInCatalogueTable (133)
NEW > N  63: bASE_RADIUS$x = 30
NEW > N  64: cAM_DEPTH$x = 28
NEW > N  65: cAM_LIFT$x = 1.65
                ...
NEW > N  66: aLPHA$x = 28
NEW > N  67: cAM_NOSE_RADIUS$x = 3
NEW > N  68: cAM_RATE$x = 0.26
                ...
```

The user can refer to the existing cam by type and EDS responds by inferring the cam's parameters, such as its lift and rate, from a table within the MCDF. Similar assumptions are used to select existing geometry for the cam roller too (not shown). At this stage assuming a specific plunger configuration enables EDS to infer the cam roller contact stress and estimate of the cam's durability (b10_LIFE$x in this case):

```
EDS> assumeEds(nUM_OF_PLUNGERS$x = 2 and pLUNGER_DIA$x = 7.5).

NEW > N  72: nUM_OF_PLUNGERS$x = 2  and  pLUNGER_DIA$x = 7.5
NEW > J  25: assumption
NEW > K  32: assertionOfConjuncts (120)
NEW > N  73: nUM_OF_PLUNGERS$x = 2
NEW > N  74: pLUNGER_DIA$x = 7.5
NEW > J  26: assertionOfConjuncts
RIP > K  32: assertionOfConjuncts (120)
NEW > K  33: valsInMulti (130)
NEW > N  75: cAM_STRESS_TSI$x = 133.277007
                ...
NEW > N  76: b10_LIFE$x = 1671.150642
                ...
```

```
!!! > New nogood set: [[1 57 61 62 69 70 72]]
!!! > Suspect Candidate for Cause is:
!!! > J  29: valsInInequality

OUT > J  29# valsInInequality
OUT > K  49# valsInInequality (140)
IN  > N   0: <false>
RIP > K  49# valsInInequality (140)
              ...
```

Unfortunately, as the EDS response lines beginning !!! > indicate, the assumption of this particular plunger configuration results in a *nogood* set of assumptions. In other words, the assumptions so far when taken together are inconsistent. More specifically, the inequality from the cAMrOLLERcONTACT MCDF would be violated: the cam's durability is not greater than that required.

To proceed with the design, the user assumes an alternative plunger configuration based on an understanding that reducing the plunger diameter will reduce the contact stress on the cam which may improve durability:

```
EDS> assumeEds(nUM_OF_PLUNGERS$x = 4 and pLUNGER_DIA$x = 5.5).

NEW > N  79: nUM_OF_PLUNGERS$x = 4  and  pLUNGER_DIA$x = 5.5
NEW > J  47: assumption
NEW > K  53: assertionOfConjuncts (120)
NEW > N  80: nUM_OF_PLUNGERS$x = 4
NEW > N  81: pLUNGER_DIA$x = 5.5
NEW > J  48: assertionOfConjuncts
RIP > K  53: assertionOfConjuncts (120)
NEW > K  54: valueConflict (150)
              ...

!!! > New nogood set: [[72 79]]
!!! > Suspect Candidate for Cause is:
```

```
!!! > J  49: valueConflict

                ...
NEW > N  82: cAM_STRESS_TSI$x = 97.736472
                ...
NEW > N  83: b10_LIFE$x = 6606.570855
                ...
```

An additional nogood set of assumptions results from the two plunger configurations being inconsistent with each other: EDS recognises this as a *valueConflict.* However, as the ATMS supports multiple-contexts, an estimate for the cam stress and durability is still derived in the context of the second plunger configuration. Note that no inconsistency results from the cam durability inequality for the second plunger configuration.

At this stage, the designer reviews progress-to-date by requesting a precis of the assumptions within the DDD. Note that we are not adding to the DDD so the output does not begin NEW > rather each line of output has the format: node (N), node number (such as 70) and node content (for example rOLLER_DIA$x = 5.98):

```
EDS> precisEds(class = assumptions).

N   1: [x] : cAMrOLLERcONTACT
N  62: cAM_TYPE$x = typeA
N  70: rOLLER_DIA$x = 5.98
N  69: rOLLER_LENGTH$x = 20.8
N  61: sAFETY_FACTOR$x = 1.0
N  57: lINE_PRESSURE$x = 720  and  rATED_SPEED$x = 1500  and
     b10_LIFE_REQ$x = 2000
N  72: nUM_OF_PLUNGERS$x = 2  and  pLUNGER_DIA$x = 7.5
N  79: nUM_OF_PLUNGERS$x = 4  and  pLUNGER_DIA$x = 5.5
aye
```

EDS provides facilities for summarising the DDD or some part of it. To clarify the results of the designer's efforts to satisfy the cam durability requirement,

consider the values of the parameters b10_LIFE$x and b10_LIFE_REQ$x within the DDD (below). As we have seen, one plunger arrangement (with node number 72) gives rise to insufficient durability whilst an alternative arrangement (node number 79) is satisfactory:

```
Node 76    Environment is ... [[1 57 62 69 70 72]]
Constraint with term  b10_LIFE$x = 1671.150642

Node 83    Environment is ... [[1 57 62 69 70 79]]
Constraint with term  b10_LIFE$x = 6606.570855

Node 58    Environment is ... [[57]]
Constraint with term  b10_LIFE_REQ$x = 2000
```

A.2.2 Prototype Adaptation with EDS

Prototype adaptation requires more work on the part of the EDS user than prototype refinement. To demonstrate prototype adaptation we assume that the DKB does not contain a single MCDF which accurately reflects the design problem space. The solution is developed by recognising that cam roller contact analysis can be treated as a specific form of the general case of Hertzian contact stress between two cylinders. Thus, the designer begins with the assumption of an instance of the cONTACT_CYLcYL module (no listing for this MCDF shown):

```
EDS> assumeEds([x]: cONTACT_CYLcYL).

NEW > N   1: [x] : cONTACT_CYLcYL
NEW > J   0: assumption
NEW > K   0: findSuperClass (125)
NEW > K   1: findDirectParts (122)
NEW > K   2: findDirectConstraints (121)
NEW > K   3: declareLabels (124)
NEW > K   4: declareDirectAttributes (123)
NEW > N   2: [x] : cONTACT_BODYbODY
```

```
NEW > J   1: findSuperClass
RIP > K   0: findSuperClass (125)
NEW > K   5: findSuperClass (125)
NEW > K   6: findDirectParts (122)
NEW > K   7: findDirectConstraints (121)
NEW > K   8: declareLabels (124)
NEW > K   9: declareDirectAttributes (123)
NEW > N   3: [x] : hETROGENEOUSoBJECT
NEW > J   2: findSuperClass
RIP > K   5: findSuperClass (125)
              ...
```

Before making an assumption for the initial requirement, the user must declare the parameters of the requirement to EDS. Recall that the DDD contains an instance of an MCDF representing two cylinders in contact so makes no mention of line pressure, rated speed and cam durability which are specific to the current problem space. Once these declarations are made, the initial requirements can be assumed:

```
EDS> assumeEds(lINE_PRESSURE$x = 720 and rATED_SPEED$x = 1500
   and b10_LIFE_REQ$x = 2000).
```

The EDS user may need to make additional parameter declarations (not shown) to adapt the contact representation to our specific cam roller problem:

```
EDS> assumeEds(pRIMARY_FORCE$x = (lINE_PRESSURE$x/10) * pi
   * ((pLUNGER_DIA$x / 2)**2)).
```

Having related the generic primary force between two cylinders in contact to the domain-specific parameters of line pressure and plunger diameter, the designer assumes an initial plunger configuration in order to derive a value of this primary force:

```
EDS> assumeEds(nUM_OF_PLUNGERS$x = 2 and pLUNGER_DIA$x = 7.5).
```

```
NEW > N  85: nUM_OF_PLUNGERS$x = 2  and  pLUNGER_DIA$x = 7.5
NEW > J  54: assumption
NEW > K  54: assertionOfConjuncts (120)
NEW > N  86: nUM_OF_PLUNGERS$x = 2
NEW > N  87: pLUNGER_DIA$x = 7.5
NEW > J  55: assertionOfConjuncts
RIP > K  54: assertionOfConjuncts (120)
NEW > K  55: valsInMulti (130)
NEW > N  88: pRIMARY_FORCE$x = 3180.862562
              ...
```

After assuming a value for alpha$x, the angle between the primary force and the contact force, the user can begin the derivation of the contact stress between the cam and roller. To do this, the user equates parameters of cylinders in contact to the desired values for the cam and roller. Note that the symbol % is used to refer to the parts of a module instance. For example, the parameter material$obja%x refers to the material of one of the constituent parts of the cONTACT_CYLcYL module instance.

At this stage we assume that obja%x represents the roller and is made of steel. A materials catalogue which forms part of this DKB, and was inherited when the MCDF instance was assumed, enables the key engineering properties of the material to be inferred by EDS:

```
EDS> assumeEds(material$obja%x = steel).

NEW > N  93: material $ obja%x  =  steel
NEW > J  66: assumption
NEW > K  65: valueInCatalogueTable (133)
NEW > K  66: valueInCatalogueTable (133)
RIP > K  66: valueInCatalogueTable (133)
NEW > N  94: e $ obja%x  =  207000
NEW > N  95: g $ obja%x  =  79300
NEW > N  96: nu $ obja%x  =  0.292
```

```
NEW > N  97: rho $ obja%x  =  0.000008
                ...
```

We can make an equivalent assumption for objb%x (not shown).

Once assumptions for the lengths of the two objects in contact have been made (not shown), EDS can infer the contact length and then the contact stress itself. Note that the contact length is inferred from a conditional constraint in the cONTACT_CYLcYL MCDF which equates the contact length to the length of the shorter of the two contact bodies (not shown):

```
EDS> assumeEds(d$objb%x = 6).

NEW > N 109: d $ objb%x  =  6
NEW > J  80: assumption
                ...
NEW > N 114: sTRESS$x = 2040.975841
                ...
```

The resulting figure for the contact stress is in the default units of the cONTACT_CYLcYL module. For this particular application, standard engineering practice is to quote cam stresses in tons per square inch (tsi). Another assumption introduces the conversion factor:

```
EDS> assumeEds(cAM_STRESS_TSI$x = 0.06475 * sTRESS$x).

NEW > N 118: cAM_STRESS_TSI$x  =  0.06475 * sTRESS$x
NEW > J 102: assumption
NEW > K 116: valsInMulti (130)
NEW > N 119: cAM_STRESS_TSI$x = 132.153186
                ...
```

Before beginning the analysis of the cam durability for the assumed plunger configuration, the EDS user employs an inequality to relate the durability requirement to the value which the designer aims to derive:

```
EDS> assumeEds(b10_LIFE$x > b10_LIFE_REQ$x).
```

In order to derive the cam life estimate the designer needs to assume a value for the contact life constant and relate the general stress frequency equation (within the cONTACT_CYLcYL MCDF) to the specific loading conditions of the cam and roller arrangement within the fuel pump.

The designer assumes a life constant equal to that used for roller bearing analysis: $\frac{10}{3}$ (assumption not shown). Similarly, the designer defines the stress frequency to be equal to the product of the number of plungers and the rated speed. To derive a real value for the cam life some empirical data is required. (Note that in the prototype refinement example this established empirical data can be represented within the cAMrOLLERcONTACT module.)

Once values for the empirical parameters have been assumed, EDS can infer the cam durability estimate (contained in datum node 132). However, this value of b10_LIFE$x is not consistent with the (assumed) inequality relating this value to the requirement. In other words, the EDS user's plunger configuration does not satisfy the requirement:

```
EDS> assumeEds(b10_LIFE_STD$x = 767).

EDS> assumeEds(sTRESS_STD$x = 135 / 0.06475).

EDS> assumeEds(nUM_PLUNGERS_STD$x = 4).

EDS> assumeEds(rATED_SPEED_STD$x = 1500).

                ...
NEW > N 132: b10_LIFE$x = 1768.206184
                ...

!!! > New nogood set: [[1 78 84 85 89 93 98 104 105 107 109 120 121 124
 126 127 128 129 130]]
!!! > Suspect Candidate for Cause is:
```

```
!!! > J 120: valsInInequality

OUT > J 120# valsInInequality
OUT > K 141# valsInInequality (140)
IN  > N   0: <false>
RIP > K 141# valsInInequality (140)
              ...
```

As in the prototype refinement example, a modification of the plunger configuration is assumed: employing four plungers instead of the initial two.

```
EDS> assumeEds(nUM_OF_PLUNGERS$x = 4 and pLUNGER_DIA$x = 5.5).

NEW > N 134: nUM_OF_PLUNGERS$x = 4  and  pLUNGER_DIA$x = 5.5
NEW > J 130: assumption
NEW > K 144: assertionOfConjuncts (120)
NEW > N 135: nUM_OF_PLUNGERS$x = 4
NEW > N 136: pLUNGER_DIA$x = 5.5
              ...

!!! > New nogood set: [[85 134]]
!!! > Suspect Candidate for Cause is:
!!! > J 132: valueConflict

              ...
NEW > N 142: sTRESS$x = 1496.715616
              ...
NEW > N 143: b10_LIFE$x = 6990.26117
              ...
```

An additional *nogood* set of assumptions results from the two plunger configurations being inconsistent with each other: EDS recognises this as a *valueConflict*. However, as the ATMS supports multiple-contexts, an estimate for the cam stress and durability is still derived in the context of the second plunger configuration.

Note that no inconsistency results from the cam durability inequality for the second plunger configuration.

At this stage, the designer reviews progress-to-date by requesting a precis of the assumptions within the DDD. Note that the list of assumptions is several times longer than was the case at the equivalent stage of the prototype refinement example (29 assumptions compared to 8). Note also that prototype adaptation requires several classes of assumption including parameter declarations, parameter values, real equations and an inequality. This is in contrast to the prototype refinement example where almost all of the designer's assumptions were directed at giving values to a single MCDF instance's parameters:

```
EDS> precisEds(class = assumptions).

N  77: Declaration of b10_LIFE_REQ$x as a parameter
N 117: Declaration of cAM_STRESS_TSI$x as a parameter
N  75: Declaration of lINE_PRESSURE$x as a parameter
N  83: Declaration of nUM_OF_PLUNGERS$x as a parameter
N 122: Declaration of nUM_PLUNGERS_STD$x as a parameter
N  82: Declaration of pLUNGER_DIA$x as a parameter
N  76: Declaration of rATED_SPEED$x as a parameter
N 123: Declaration of rATED_SPEED_STD$x as a parameter
N   1: [x] : cONTACT_CYLcYL
N  89: aLPHA$x = 28
N 127: b10_LIFE_STD$x = 767
N 118: cAM_STRESS_TSI$x  =  0.06475 * sTRESS$x
N 105: d $ obja%x  =  5.98
N 109: d $ objb%x  =  6
N 104: l $ obja%x  =  20.8
N 107: l $ objb%x  =  28
N 121: lIFE_CONST$x = 3.333333
N  93: material $ obja%x  =  steel
N  98: material $ objb%x  =  steel
N 129: nUM_PLUNGERS_STD$x = 4
N  84: pRIMARY_FORCE$x   =   0.07854  *  lINE_PRESSURE$x  *
```

```
     pLUNGER_DIA$x ** 2
N 130: rATED_SPEED_STD$x = 1500
N 124: sTRESS_FREQ$x  =  nUM_OF_PLUNGERS$x * rATED_SPEED$x
N 126: sTRESS_FREQ_STD$x  =  nUM_PLUNGERS_STD$x * rATED_SPEED_STD$x
N 128: sTRESS_STD$x = 2084.942085
N 120: b10_LIFE$x > b10_LIFE_REQ$x
N  78: lINE_PRESSURE$x = 720  and  rATED_SPEED$x = 1500  and
     b10_LIFE_REQ$x = 2000
N  85: nUM_OF_PLUNGERS$x = 2  and  pLUNGER_DIA$x = 7.5
N 134: nUM_OF_PLUNGERS$x = 4  and  pLUNGER_DIA$x = 5.5
aye
```

EDS provides facilities for summarising the DDD or some part of it. To clarify the results of the designer's efforts to satisfy the cam durability requirement, consider the values of the parameters b10_LIFE$x and b10_LIFE_REQ$x within the DDD (below). As we have seen, one plunger arrangement (with node number 85) gives rise to insufficient durability whilst an alternative arrangement (node number 134) is satisfactory:

```
Node 132    Environment is ... [[1 78 84 85 89 93 98 104 105 107 109
    121 124 126 127 128 129 130]]
Constraint with term  b10_LIFE$x = 1768.206184

Node 143    Environment is ... [[1 78 84 89 93 98 104 105 107 109 121
124 126 127 128 129 130 134]]
Constraint with term  b10_LIFE$x = 6990.26117

Node 79    Environment is ... [[78]]
Constraint with term  b10_LIFE_REQ$x = 2000
```

One likely consequence of the decision to adopt four smaller plungers rather than two larger is that there may have been an undesirable impact on the capability of the design to satisfy a fuelling requirement. Thus, for both the prototype refinement and prototype adaptation examples, the analysis of the cam roller contact necessarily represents only one part of a larger design exploration process.

Appendix B

Derivation of the Loss Function

This appendix contains a derivation of the loss function which relates the change in a characteristic's worth with the deviation of the characteristic from its nominal value. In [Taguchi, 1986; Taguchi *et al*, 1989] a characteristic is always associated with a product and worth is similarly always evaluated in monetary terms. Arguably, neither of these restrictions is necessary and the following description retains generality where possible.

Loss is defined to occur whenever a characteristic (by convention denoted by y) deviates from its nominal or target value (by convention denoted by m). Thus, loss resulting from this deviation of a characteristic equals zero when $y = m$ and increases as y moves away from m either upward or downward.

Denote the function which relates loss to a characteristic y as $\mathrm{L}(y)$ and expand this function about the nominal value m as a Taylor series:

$$\mathrm{L}(y) = \mathrm{L}(m + y - m) \tag{B.1}$$

$$\mathrm{L}(y) = \mathrm{L}(m) + \frac{\mathrm{L}'(m)}{1!}(y - m) + \frac{\mathrm{L}''(m)}{2!}(y - m)^2 + \ldots \tag{B.2}$$

By definition, $\mathrm{L}(y) = 0$ when $y = m$ as the characteristic loss is zero at the target value. Similarly, as we defined the loss function as one which increases with deviations away from the target value, then clearly the target value represents a

minimum and the first derivative of the loss function at this point, $L'(m)$, equals zero:

$$L(y) = \frac{L''(m)}{2!}(y-m)^2 + \ldots \tag{B.3}$$

If we neglect terms in $(y-m)$ higher than order 2 and substitute a constant c for the value of the second derivative at m:

$$L(y) = c(y-m)^2 \tag{B.4}$$

In order to make use of this loss function, a value for c is required. Assume that the loss A is known to be associated with the characteristic deviation Δ. It follows that, by substitution in Equation B.4:

$$A = c\Delta^2 \tag{B.5}$$

By rearranging Equation B.5 we derive a value for loss function constant c which can be substituted into Equation B.4 to provide a usable version of the loss function:

$$L(y) = \frac{A}{\Delta^2}(y-m)^2 \tag{B.6}$$

This relationship, which underlies Taguchi's approach to quality engineering including tolerance design is shown in Figure B–1:

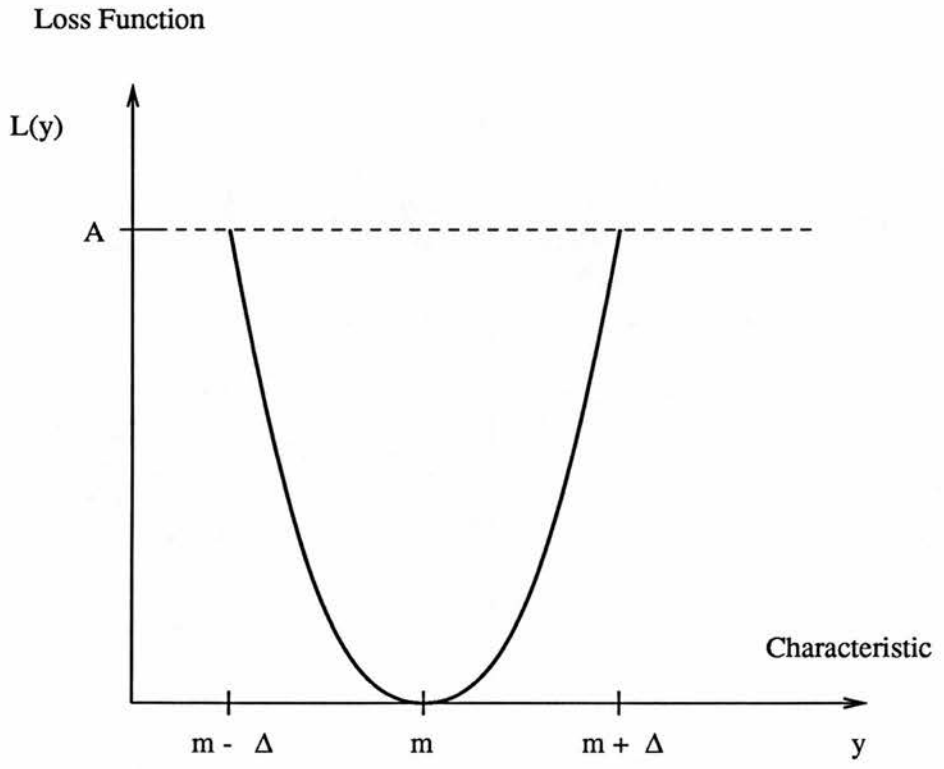


Figure B–1: Relationship between Characteristic Value and Loss

References

Arbab, Farhad. (1982). *Requirements and Architecture of a CAM Oriented CAD System for Design and Manufacture of Mechanical Parts.* Unpublished Ph.D. thesis, University of California, Los Angeles.

Barrett, R., Ramsay, A. and Sloman, A. (1985). *POP-11: A Practical Language for Artificial Intelligence.* Ellis Horwood Limited, first edition.

Bowen, J. and Bahler, D. (1992). Supporting multiple perspectives: a constraint-based approach to concurrent engineering. In Gero, J., (ed.), *Artificial Intelligence in Design '92.*

Brooks, R. A. (1981). Symbolic Reasoning Among 3-D Models and 2-D Images. *Artificial Intelligence*, 17:285–348.

Brooks, R. A. (1984). Symbolic error analysis and robot planning. *International Journal of Robotics Research*, 1.

BSI/Hutchinson. (1984). *Manual of British Standards in ENGINEERING DRAWING AND DESIGN*, first edition.

BSI Education. (1985). *An Introduction to the Tolerancing of Functional Length Dimensions*, first edition, Document reference: PP 7309.

Carter, Ian M. (1990). Applications and prospects for AI in mechanical engineering design. *The Knowledge Engineering Review*, 3(5):167–179.

Lucas CAV Limited. (1979). *Rotary Pump Cam Stress and Life Calculations*, Unpublished Proving Lab Note No. 573A.

Chang, T.-C. and Wysk, R. A. (1985). *An Introduction to Automated Process Planning Systems.* Prentice-Hall, first edition.

Chang, T.-C. (1990). *Expert Process Planning for Manufacturing.* Addison-Wesley, first edition.

Clocksin, W. F. and Mellish, C. S. (1984). *Programming in Prolog.* Springer-Verlag, second edition.

Corner, D. F., Ambler, A. P. and Popplestone, R. J. (1983). Reasoning about the Spatial Relationships derived from a RAPT Program for Describing Assembly by Robot. In *Proceedings of the 8th International Joint Conference on Artificial Intelligence*, Karlsruhe.

Coyne, Robert F. (1989). *Planning in Design Synthesis: Abstraction-Based LOOS (ABLOOS).* Technical Report EDRC-48-14-89, Carnegie Mellon University, Engineering Design Research Center.

de Kleer, J. (1984). Choices without backtracking. In *Proceedings of AAAI-84*, pages 79–85.

Descotte, Yannick and Latombe, Jean-Claude. (1981). GARI: A problem solver that plans how to machine mechanical parts. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence*, pages 766–772.

Dixon, John R. (1986). Artificial intelligence and design: a mechanical engineering view. In *Proceedings of AAAI-86*, pages 872–877.

Duffy, A. (1987). Bibliography — artificial intelligence in design. *Artificial Intelligence in Engineering*, 2(3):173–179.

Durrant-Whyte, Hugh F. (1986). Concerning Uncertain Geometry in Robotics. In *Proceedings of Geometric Reasoning Workshop, Oxford.*

Faux, I. D. and Pratt, M. J. (1979). *Computational Geometry for Design and Manufacture.* Ellis Horwood.

Finger, Susan and Dixon, John R. (1989a). A review of research in mechanical engineering design. Part I: Descriptive, prescriptive, and computer-based models of design processes. *Research in Engineering Design*, 1:51–67.

Finger, Susan and Dixon, John R. (1989b). A review of research in mechanical engineering design. Part II: Representations, analysis, and design for the life cycle. *Research in Engineering Design*, 1:121–137.

Fischer, G. and Nakakoji, K. (1991). Empowering designers with integrated design environments. In Gero, J., (ed.), *Artificial Intelligence in Design '91.*

Fleming, Alan. (October 1987). *Analysis of Uncertainties and Geometric Tolerances in Structures of Parts.* Unpublished Ph.D. thesis, Edinburgh University, Department of Artificial Intelligence.

Forbus, K. D. and de Kleer, J. (1988). Focusing the ATMS. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 193–198.

Gero, J. S. and Rosenman, M. A. (1990). A conceptual framework for knowledge-based design research at Sydney University's design computing unit. *Artificial Intelligence in Engineering*, 5(2):65–77.

Gero, J. S., Maher, M. L. and Zhang, W. (1988). Chunking structural design knowledge as prototypes. In *3rd International Conference on Applications of Artificial Intelligence in Engineering Problems.*

Groover, Mikell P. and Zimmers, Emory W. Jr. (1984). *CAD/CAM: Computer-aided design and manufacturing.* Prentice-Hall International, first edition.

Hayes-Roth, B. (1985). Blackboard architectures for control. *Journal of Artificial Intelligence*, 26:251–321.

Herbert, P. J., Hinde, C. J., Bray, A. D., Launders, V. A., Round, D. and Temple, D. M. (1990). Feature recognition within a truth maintained process planning system. *International Journal of Computer Integrated Manufacture.*

Hillyard, R. C. and Braid, I. C. (1978). Characterizing non-ideal shapes in terms of dimensions and tolerances. *ACM Computer Graphics*, 12(3):234–238.

Hinde, C. J., Bray, A. D., Herbert, P. J., Launders, V. A. and Round, D. (1989). A truth maintained approach to process planning. In *4th International Conference on Applications of Artificial Intelligence in Engineering Problems.*

Hoffman, Peter. (1982). Analysis of tolerances and process inaccuracies in discrete part manufacturing. *Computer-Aided Design*, 14(2):83–88.

Howe, A. E., Cohen, P. R., Dixon, J. R. and Simmons, M. K. (1986). Dominic : A Domain-Independent Program for Mechanical Engineering Design. *Artificial Intelligence*, 1(1).

Institution of Electrical Engineers, Computing and Control Division. (1990). *Colloquium on "The Alvey Design to Product Demonstrator Project".* Digest No. 1990/119.

Ingham, P. C. (1980). *A Computer-Aided Tolerance System.* Technical report, City of Birmingham Polytechnic.

International Organisation for Standardisation, Geneva, Switzerland. (1982). *ISO/R1101: Technical Drawings – Tolerances of Form and of Position.*

Ito, Y., Shinno, H. and Saito, H. (1988). A proposal for CAD/CAM interface with expert systems. *Robotics & Computer-Integrated Manufacture*, 4(3).

Joshi, S., Chang, T. C. and Liu, C. R. (1986). Process planning formalization in an AI framework. *Artificial Intelligence*, 1(1).

Juster, N. P. (1992). Modelling and representation of dimensions and tolerances: a survey. *Computer-Aided Design*, 24(1):3–17.

Kim, C., O'Grady, P. and Young, R. E. (1992). A large scale, multiple constraint network system for design for testability for printed wiring boards. In Gero, J., (ed.), *Artificial Intelligence in Design '92.*

Latombe, J.-C. (1976). *Artificial Intelligence in Computer-aided Design: The 'Tropic' System.* Technical Note 125, Stanford Research Institute, Artificial Intelligence Center.

Light, Robert and Gossard, David. (1982). Modification of geometric models through variational geometry. *Computer-Aided Design*, 14(4):209–214.

Logan, B. and Smithers, T. (1989). *The Role of Prototypes in Creative Design.* DAI Research Paper 453, Department of Artificial Intelligence, Edinburgh University.

Logan, B., Millington, K. and Smithers, T. (1991). Being economical with the truth: assumption-based context management in the Edinburgh Designer System. In Gero, J., (ed.), *Artificial Intelligence in Design '91.*

MacCallum, K. J., Duffy, A. and Green, S. (October 1985). An Intelligent Concept Design Assistant. In *IFIP W.G. 5.2 Working Conference on Design Theory for CAD*, pages 233–249.

Mayer, Richard J., Su, Chuan-Jun and Keen, Arthur K. (1992). An integrated manufacturing planning assistant – IMPA. *Journal of Intelligent Manufacturing*, 3:109–122.

Mullins, Scott and Rinderle, James R. (1991). Grammatical approaches to engineering design, Part I: An introduction and commentary. *Research in Engineering Design*, 2:121–135.

Murthy, Seshashayee S. and Addanki, Sanjaya. (1988). PROMPT: An Innovative Design Tool. In *Proceedings of AAAI-88*, pages 637–642.

Nau, D. S. (August 1987). Hierarchical abstraction for process planning. In Sriram, D. and Adey, R. A., (eds.), *2nd International Conference on Applications of Artificial Intelligence in Engineering Problems.*

Orelup, M. F., Cohen, P. R., Dixon, J. R. and Simmons, M. K. (1988). Dominic II: Meta-Level Control in Iterative Redesign. In *Proceedings of AAAI-88*, pages 79–85.

O'Shea, Tim and Eisenstadt, Marc. (1984). *Artificial Intelligence, Tools, Techniques, and Applications*, chapter 4, pages 110–136. Harper and Row.

Pahl, G. and Beitz, W. (1988). *Engineering Design – A Systematic Approach.* Design Council, first edition.

Parkinson, D. B. (1984). *Tolerancing of Component Dimensions in CAD.* Technical report, Department of Mechanical Engineering, University of Liverpool.

Rehg, J., Elfes, A., Talukdar, S., Woodbury, B., Eisenberger, M. and Edahl, R. H. (1988). Design Systems Integration in CASE. In Rychener, Michael D., (ed.), *Expert Systems for Engineering Design*, pages 279–301. Academic Press.

Requicha, A. A. G. and Chan, S. C. (October 1985). *Representation of Geometric Features, Tolerances and Attributes in Solid Modellers Based on Constructive Solid Geometry.* Technical Memorandum No. 48, Production Automation Project, University of Rochester.

Requicha, A. A. G. and Vandenbrande, J. (1988). Automated systems for process planning and part programming. In Kusiak, A., (ed.), *Artificial Intelligence: implications for computer-integrated manufacturing*, chapter 8. IFS (Publications) Ltd.

Requicha, A. A. G. (August 1983). *Representation of Tolerances in Solid Modelling: Issues and Alternative Approaches.* Technical Memorandum No. 41, Production Automation Project, University of Rochester.

Requicha, A. A. G. (March 1983). *Toward a Theory of Geometric Tolerancing.* Technical Memorandum No. 40, Production Automation Project, University of Rochester.

Robertson, Andy. (September 1990). The Alvey 'Design to Product' Demonstrator Project: A User's View. In *Expert Systems 90 – Papers from Application Stream.* British Computer Society's Specialist Group on Expert Systems.

Rooney, Joe and Steadman, Philip, (eds.). (1987). *Principles of Computer-Aided Design.* Pitman, first edition.

Rychener, M.D., Farinacci, M.L., Hulthage, I. and Fox, M.S. (September 1986). *Integration of Multiple Knowledge Sources in ALADIN, An Alloy Design System.* Technical Report EDRC-05-04-86, Carnegie Mellon University, Engineering Design Research Center.

Shigley, J.E. (1986). *Mechanical Engineering Design.* McGraw-Hill Book Company, New York, first metric edition.

Simon, Herbert A. (1969). *The Sciences of the Artificial.* The MIT Press, first edition.

Smithers, Tim. (November 1985). The Alvey Large Scale Demonstrator Project 'Design to Product'. In *Artificial Intelligence in Manufacturing: Key to Integration?*, Gottlieb Dutweiler Institute, Zurich.

Smithers, Tim. (1989). AI-based Design v Geometry-based Design, or, Why Design Cannot be Supported by Geometry Alone. *Computer Aided Design*, 21(3):141–150.

Smithers, T., Conkie, A., Doheny, J., Logan, B., Millington, K. and Tang, Ming Xi. (1989). *Design as Intelligent Behaviour: An AI in Design Research Programme.* DAI Research Paper 426, Edinburgh University, Department of Artificial Intelligence.

Sussman, G.J. (June 1977). *Electrical Design: A Problem for Artificial Intelligence Research.* AI Memo 425, Massachusetts Institute of Technology, AI Laboratory.

Swift, K. G., Matthews, A. and Runciman, C., (1984). *Knowledge Based Systems in Engineering Design.*

Taguchi, Genichi. (1986). *Introduction to Quality Engineering.* Asian Productivity Organization, first edition.

Taguchi, G., Elsayed, E. A. and Hsiang, T. C. (1989). *Quality Engineering in Production Systems.* McGraw-Hill Book Company, first edition.

Tong, C. (August 1990). *Knowledge-based Design as an Engineering Science: The Rutgers AI/Design Project.* Technical Report WP-TR-167, Rutgers University, Laboratory for Computer Science Research.

Voelcker, H. B., Requicha, A. A. G. and Conway, R. W. (1988). Computer applications in manufacturing. *Annual Review of Computer Science*, 3:349–387.

Wang, H-P. (August 1987). A knowledge-based computer-aided process planning system. In Sriram, D. and Adey, R. A., (eds.), *2nd International Conference on Applications of Artificial Intelligence in Engineering Problems.*

Westerberg, A., Grossman, I., Talukdar, S., Prinz, F., Fenves, S. and Maher, M. L. (1989). Applications of AI in design research at Carnegie Mellon University's EDRC. In *4th International Conference on Applications of Artificial Intelligence in Engineering.*

Winograd, Terry and Flores, Fernando. (1986). *Understanding Computers and Cognition.* Addison-Wesley Publishing Company, Inc., Third Printing, August 1988.