

BLDSC no:- DX 210836



Pilkington Library

Author/Filing Title HOUSELL, -M.S.

Accession/Copy No.

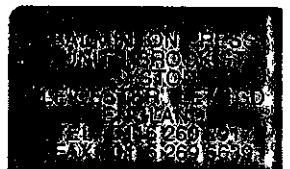
Vol. No.

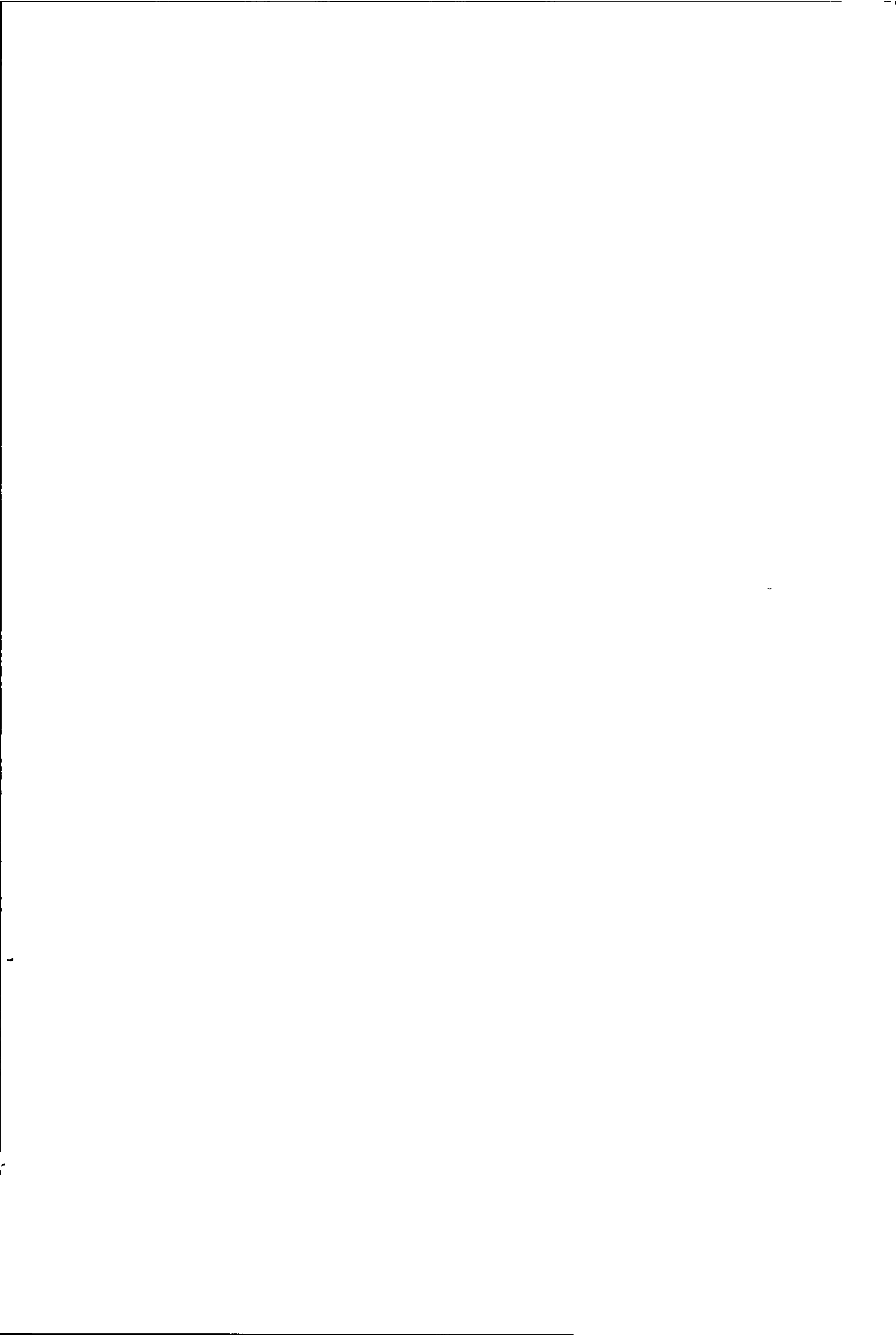
Class Mark T

14 JAN 2000

LOAN COPY

0401991709





**FEATURE-BASED VALIDATION REASONING
FOR INTENT-DRIVEN ENGINEERING DESIGN**

by

Marcelo da Silva Hounsell


A DOCTORAL THESIS

Submitted in partial fulfilment of the requirements for the award of

Doctor of Philosophy of Loughborough University

July-1998

© by Marcelo da Silva Hounsell (1998)

	Loughborough University
P.	May
Date	Apr 99
Clas	
Acc No.	020199170

K0645116

ACKNOWLEDGEMENTS

I am deeply indebted to my supervisor, Professor Keith Case, and I would like to express my sincere gratitude to him for sharing some of his knowledge with me, for his expert guidance, for his friendship, and for his endless help with my attempts to grasp the idiosyncrasies of the English language.

Acknowledgements are due to the Brazilian Federal Agency for Post-Graduate Education, CAPES, for providing financial support for this research, and to the State University of Santa Catarina (Brazil), UDESC, for granting me study leave.

Special thanks must go to my parents, Eduardo and Regina, who have taught me the important things in life, and who have suffered with the burden of the long distance of our separation.

Special thanks must also go to my mother-in-law, Dona Cata, who came and helped my wife and me during a very important period of our lives.

I would like to thank all my friends of the 'Brazilian Community' in Loughborough. Their support and friendship made this work seem less laborious.

To Renata and Carolina Hounsell,
my beloved wife and daughter.

ABSTRACT

Feature based modelling represents the future of CAD systems. However, operations such as modelling and editing can corrupt the validity of a feature-based model representation. Feature interactions are a consequence of feature operations and the existence of a number of features in the same model. Feature interaction affects not only the solid representation of the part, but also the functional intentions embedded within features. A technique is thus required to assess the integrity of a feature-based model from various perspectives, including the functional intentional one, and this technique must take into account the problems brought about by feature interactions and operations. The understanding, reasoning and resolution of invalid feature-based models requires an understanding of the feature interaction phenomena, as well as the characterisation of these functional intentions. A system capable of such assessment is called a feature-based representation validation system.

This research studies feature interaction phenomena and *feature-based designer's intents* as a medium to achieve a feature-based representation validation system.

It was found that feature interaction classifications available in the literature are strongly oriented towards the feature recognition approach and are mainly inappropriate to design-by-features systems. A feature interaction classification and identification mechanism is thus proposed. In addition, a taxonomy of *designer's intents* is proposed that makes explicit many of the expected behaviours behind the use of features in a representation for specific applications. The binding process that relates feature interactions to *designer's intents* allows the validity assessment of the representation and also the identification of operations that contribute to the revalidation of the representation. This bounding process leads to a reasoning mechanism that performs feature validation and is driven by *designer's intents*, and, therefore, was baptised **FRIEND** (Feature-based validation Reasoning for Intent-driven ENgineering Design).

TABLE OF CONTENTS

ACKNOWLEDGEMENTS	III
ABSTRACT	V
TABLE OF CONTENTS	VI
LIST OF ABBREVIATIONS	XIV
1. INTRODUCTION	1
1.1 CAD Systems	1
1.2 Feature-based CAD systems	3
1.3 Commercial Feature-based CAD Systems	5
1.4 The Validation Problem	5
1.5 Distinct Validation Aspects	6
1.5.1 Model Validation	7
1.5.2 Entity Elicitation	8
1.5.3 Representation Validation	15
1.5.4 Examples of the Validation Aspects	18
1.6 Objectives	19
1.6.1 Motivation	19
1.6.2 Objectives Statement	20
1.6.3 Specific Objectives	21
1.6.4 Generic Scope	22

1.7 Resulting Publications	22
1.8 Structure of the Thesis.....	24
1.9 Summary	25
2. LITERATURE REVIEW.....	26
2.1 Feature Definitions.....	26
2.2 Types of Features	29
2.3 Feature-based Applications.....	33
2.4 Feature Representations	34
2.4.1 B-Rep, CSG, Hybrid Schemes and Enhanced Representations	34
2.4.2 Volume and Surface Features.....	35
2.5 Feature Implementation Approaches.....	36
2.6 System Approaches	37
2.6.1 Design-by-Features (DbF) Approach	38
2.6.2 Feature Recognition (FeR) Approach	39
2.6.3 Hybrid Design-by and Recognise Features Approach (HDR).....	41
2.7 Related Work on Representation Validation.....	43
2.8 Summary	59
3. FEATURE-BASED VALIDATION	60
3.1 Facets of Validation.....	60
3.1.1 GSM validation	60
3.1.2 GSM-like Validation.....	61
3.1.3 Constraint Validation	61
3.1.4 Manipulation Restrictions	61
3.1.5 Associative Validation	62
3.1.6 Intersection Validation	62

3.1.7 Rule-based Validation	62
3.1.8 Consistency Among Representations	62
3.1.9 Feature's Topological Validation	63
3.1.10 Validation via Recognition.....	63
3.2 Representation Validation.....	63
3.2.1 Two Level Validation.....	63
3.2.2 Conceptual Feature Validation	65
3.2.3 Validation at Feature Class Level.....	66
3.3 Feature Validation System	67
3.3.1 Domain Characterisation	67
3.3.2 Validity Conditions	68
3.3.3 Operations Characterisation	68
3.3.4 The Framework	69
3.4 Detailed Research Scope.....	69
3.5 What to Validate ? Feature-based Designer's Intents	70
3.5.1 Definition	72
3.5.2 Which FbDI's ? Volumetric FbDI's.....	73
3.5.3 What are FbDI's ? The Essence	81
3.5.4 The Actual VDI's	82
3.6 Example of Conceptual MFI Reasoning	83
3.7 Composing a Feature Validation System	84
3.8 Summary	85
4. AN INTENT-DRIVEN APPROACH.....	86
4.1 Conceptual Validation and Beyond.....	86
4.1.1 Feature-based Designer's Intents Elicitation.....	88
4.2 Designer's Intents Elicitation Criteria	88
4.3 Designer's Intents Classification.....	89

4.3.1 Morphological Functional FbDI (MFI)	89
4.3.2 Theoretical Functional FbDI (TDI)	90
4.3.3 Relational Functional FbDI (RDI).....	91
4.4 Designer's Intents Taxonomy.....	92
4.4.1 Morphological FbDI.....	92
4.4.2 Parametric FbDI	92
4.4.3 Geometric FbDI's and Application-Oriented Intents	94
4.5 A FbDI Taxonomy.....	103
4.6 "Intenturization" Validation.....	105
4.7 Summary	106
5. FEATURE-BASED INTERACTION	107
5.1 The Need for a Broad Coverage.....	107
5.2 Terminology	109
5.3 Related Work on Feature Interaction	110
5.3.1 Types of Feature Interactions	111
5.3.2 Some Feature Interaction Classifications	112
5.3.3 A Discussion on Existing Classifications.....	119
5.4 The Classification Framework.....	121
5.4.1 Entities and Levels	121
5.4.2 Queries to the Underlying GSM.....	122
5.4.3 The Identification Process	123
5.4.4 The Basic Framework	126
5.4.5 The Complete Classification Tree.....	128
5.4.6 Special Meanings and a Few Exceptions	128
5.5 Summary	131
6. OPERATING FEATURE MODELS	132

6.1 Operations and Validation	132
6.2 Operations Classification.....	137
6.2.1 Analysis Operations.....	137
6.2.2 Manipulation Operations	137
6.2.3 Setup Operations	138
6.2.4 The Operations Classification	139
6.3 Manipulation Operations	139
6.3.1 Modelling Operations.....	139
6.3.2 Editing Operations.....	141
6.3.3 Revalidation Operations	142
6.4 A Minimum Set of Operations	148
6.5 Summary	149
7. VALIDITY CONDITIONS	150
7.1 Organising the Reasoning.....	150
7.2 Invalidity Tests	152
7.2.1 The Validation Process.....	152
7.3 Organisation	155
7.3.1 Reasoning Aspects	155
7.3.2 Reasoning Sets	156
7.3.3 Priority	158
7.4 Intents Management	159
7.4.1 Verification Statements	160
7.4.2 Enrichment Statements.....	161
7.4.3 Update Statements.....	162
7.5 Active, Inactive and Intentional Status	163
7.6 Intent Management Priority.....	165

7.7 Understanding the Reasoning Organisation.....	166
7.8 Summary	168
8. IMPLEMENTATION.....	169
8.1 A Prototype Implementation.....	169
8.2 Resources.....	170
8.2.1 WxCLIPS	170
8.2.2 Microstation	172
8.3 Modules	172
8.3.1 FRIEND-VIEW ..	172
8.3.2 FRIEND-KBS.....	173
8.4 Data Structures.....	178
8.4.1 Representing Intents and Interactions.....	178
8.4.2 Representing Features .	182
8.5 Intents Management Implementation	188
8.6 Feature Interaction Identification Implementation	189
8.7 Feature Operations Implementation	189
8.8 Priority Implementation	190
8.9 Reasoning Sets Examples.....	191
8.9.1 Simply Geometrical Reasoning	191
8.9.2 Simply Volumetrical Reasoning.....	192
8.9.3 Simply Labelling Reasoning	193
8.9.4 Complex Reasoning	195
8.10 Intents Management Examples.....	198
8.10.1 Experience-based Guided Enrichment	198
8.10.2 Blind Enrichment	200
8.10.3 Verification.....	201

8.10.4 Inheritance-based Guided Enrichment	203
8.11 Final Remarks on The Implementation	204
8.12 Summary	205
9. TEST CASES	206
9.1 Introduction	206
9.2 Standard Orientation.....	207
9.3 Labelling.....	209
9.4 Valid Part Description	211
9.5 Morphological Reasoning Test.....	214
9.6 Thin Wall Test Cases	216
9.7 Chang's Part	218
9.8 A Lost Intention?.....	220
9.9 Information for Process Planning.....	225
9.9.1 Visualisation.....	225
9.9.2 Redesign	229
9.10 Edinburgh Composite Component	232
9.11 A Comparison Problem	236
9.12 Summary	237
10. DISCUSSION.....	238
10.1 The Validation Framework	238
10.2 Intent-Driven versus Constraint-driven Only Approaches.....	239
10.3 Use of the Elicitation Process	241

10.4 Designer's Intents	241
10.5 Feature Interactions	243
10.6 Feature Operations	244
10.7 The Reasoning Priority	247
10.8 Intents Management	248
10.9 The Implementation	249
11. CONCLUSION	251
11.1 Summary of the Thesis	251
11.2 Contributions	254
11.2.1 Clarifications	254
11.2.2 Effective Capture of Designer's Intents	255
11.2.3 The Validation Framework	255
11.2.4 Information Detailing (The Process).....	257
11.2.5 Classifications and Taxonomies (The Products).....	258
11.2.6 Priority Arrangement.....	259
11.2.7 Added Development Formalism	260
11.2.8 A Comparison Framework	260
11.3 Further Work and Future Research	261
11.3.1 Extensions	261
11.3.2 Expansions	263
11.4 Final Remarks	271
12. REFERENCES	272

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
AOI	Application-Oriented Designer's Intent
Bbox	Bounding Box
B-rep	Boundary Representation
BI	Boundary Interaction
CAD	Computer Aided Design
CAE	Computer Aided Engineering
CAM	Computer- Aided Manufacturing
CAPC	Computer Aided Planning and Control
CAPP	Computer Aided Process Planning
CE	Concurrent Engineering
CLIPS	C Language Integrated Production System
CMM	Coordinate Measuring Machine
CSG	Constructive Solid Geometry
DbF	Design-by-Features
DFA	Design for Assembly
DFFix	Design for Fixturing
DFM	Design for Manufacturability
DFMould	Design for Mouldability

DP	Design Process
DSG	Destructive Solid Geometry
EAD	External Access Direction
ECTO	Extended CSG Tree of Features
FAB	Feature's Associated Boundary
FAG	Feature-Dependency Graph
FAS	Feature's Associated Surfaces or Faces
FAV	Feature Associated Volume
FbDI	Feature-based Designer's Intent
FBM	Feature-based Modelling
FDG	Feature-Dependency Graph
FeR	Feature Recognition
FI	Facial Interaction
FIG	Feature Interaction Graph
FN	Feature Nature
FPS	Feature Producing Surfaces
FPV	Feature Producing Volume
FRIEND	Feature-based validation Reasoning for Intent-driven Engineering Design
FV	Feature Volume

GDI	Geometric Designer's Intent
GSM	Geometric Solid Modelling
GT	Group Technology
HDR	Hybrid Design-by and Recognise Features
ICAD	Intelligent CAD
KBS	Knowledge-based System
MFI	Morphological Functional FbDI's
MMF	Manufacturing Motion Feature
NA	Non Applicable
PDI	Parametric Designer's Intent
PDM	Product Data Modelling
RDI	Relational Designer's Intent
SE	Semantic Entities
TDI	Theoretical Functional Designer's Intent
VDI	Volumetric Designer's Intent
VI	Volumetric Interaction

1. INTRODUCTION

1.1 CAD SYSTEMS

Computer Aided Design (CAD) systems are considered an essential tool for detailed geometric design allowing a high level of flexibility, efficiency and quality. Traditional CAD systems use low level entities such as *vertices*, *edges*, and *faces* (see Figure 1-1), as well as low-level operators such as *move a vertex*, *create an arc*, *delete an edge* and *insert a face* for detailing a geometric design.

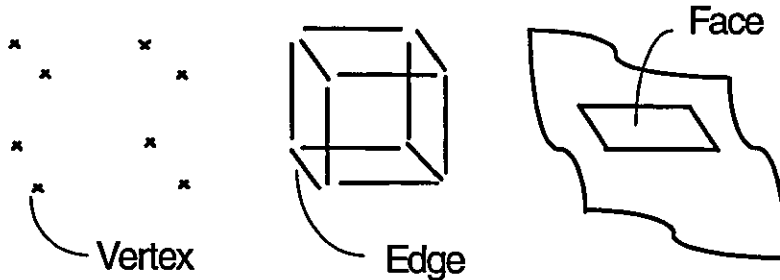


Figure 1-1: Some Entities of Traditional CAD Systems.

More recent CAD systems based on Geometric Solid Modelling (GSM) use solid primitives of various shapes such as *spheres*, *cones* and *cylinders* (Figure 1-2) that can be combined using Boolean operators such as *union*, *intersection* and *difference*.

Although GSM represents an important, and nowadays widespread, improvement over 2D and 2.5 D computer-aided drafting systems they have some unattractive factors when being considered as the medium for integration of CAD systems with other computer-aided engineering-related activities

(such as process planning - Mantyla89 - and tolerancing analysis - Duan89). Such limitations include:

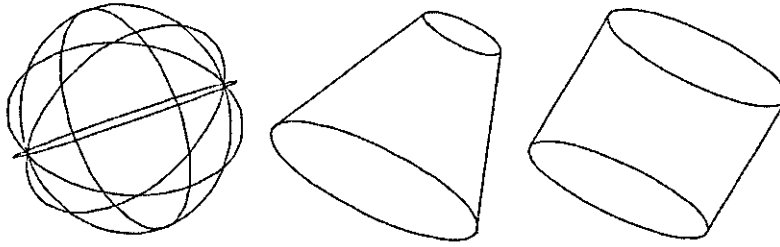


Figure 1-2: Entities of GSM-based CAD Systems.

- difficulty in interpreting geometric information from the point of view of manufacturing;
- difficulty in providing associated (non-geometrical) information needed for process planning;
- being large and complex data structures themselves, GSM representations are not attractive for handling extra attribute information which can be complex and voluminous;
- because GSM represents one level of information (geometry) it has been considered to be single-level. Thus, it is difficult to distinguish between essential functional aspects of the shape to be used by the application and other non-essential aspects;
- traditional primitives in GSM modellers are not convenient for defining geometric tolerancing and manufacturing specifications.

Furthermore, the integration or even interfacing of current CAD systems with other activities such as engineering (CAE), process planning (CAPP), manufacturing (CAM) and production control (CAPC) has been shown to be a difficult task because current CAD systems are incapable of capturing non-geometric aspects of the designer's intent such as tolerances, part relationships or surface finish. (Nnaji93, Stroud93, Marefat93b). In addition, more abstract design activities such as conceptual design, generation of design alternatives,

reuse and reasoning on design procedures and capturing the functionality of a product are just impossible (Henderson93, Taylor96).

These limitations have generated a research area, which takes their resolution as its main objectives and is known as Feature-based Modelling.

1.2 FEATURE-BASED CAD SYSTEMS

Feature-based Modelling (FBM) systems enhance existing CAD environments through the use of more meaningful entities, ease of use and facilitate integration with other computer-aided systems within the manufacturing context because they subsume extra non-geometric semantics.

FBM systems use entities, called *features*, that are closer to the designer's own vocabulary such as *holes*, *slots* and *steps* (Figure 1-3) and are considered the means of incorporating knowledge of the form, behaviour, function and related manufacturing processes into a single representation (ElMaraghy93a).

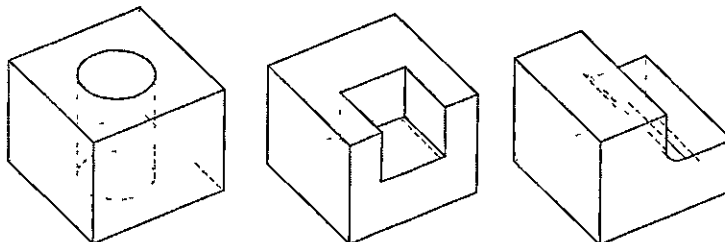


Figure 1-3: Entities of Feature-based Modelling Systems.

Feature-based Modelling has already become an enabling product modelling technique for a better integration of CAD systems and other engineering-related analyses. Its close integration with specific disciplines such as manufacturing and assembly has helped develop approaches such as Design-for-Manufacturing (DFM) and Design-for-Assembly (DFA). It has even been asserted that "one of the most popular approaches for manufacturing involves features, or recurring shapes with some fixed engineering significance" (Mantyla96).

The possibility of capturing designer's intent (see discussion on this in section 3.5), as well as geometric and topological information, also helps mediate

multi-disciplinary product development towards a Concurrent Engineering (CE) approach (Lim95).

Furthermore, FBM plays an important role in helping the development of Intelligent CAD (ICAD) systems where, with the help of Artificial Intelligence (AI) techniques, not only the object, but also the Design Process, can be modelled and manipulated (Ohsuda89, Dixon90, Nielsen91).

Design-by-Features (DbF) is one approach for implementing Feature-based CAD systems and offers the designer a library of features to be used to represent the desired component. DbF systems are distinct from the Feature Recognition (FeR) approach where features are 'discovered' after a session using traditional or GSM-based CAD systems. Current thinking is that elements of each approach (traditional CAD, GSM CAD, DbF and, FeR) can be usefully combined into a DbF-like system producing a much more useful and flexible system (see section 2.6.3).

The power of feature models in manufacturing applications is based on associating feature types with manufacturing process models (Mäntylä96). This assertion emphasises a need for FBM systems to produce correct model representations. Correct representations can be used immediately by downstream applications without the need for further filtering of errors or misrepresentations introduced by the use of the feature-based modelling technique.

The main advantages of using features include (Ovtcharova92):

- a feature vocabulary is more natural for expressing the product when compared with a purely geometric one;
- there is a possibility of using features as a basis for modelling product information in different phases such as design, analysis, process planning and manufacturing;
- the use of features can lead to an increase in designer's productivity and cost effectiveness.

1.3 COMMERCIAL FEATURE-BASED CAD SYSTEMS

Feature-based modelling has been reported to have been incorporated in (parametric) commercial CAD systems (Shah91, Rosen93). These commercial systems include Microstation 95, Pro/ENGINEER, UNIGRAPHICS (Shah91, Mitchell96), BRAVO, CADD5 and I-DEAS (Lim95).

However, these implementations have suffered severe criticism:

- “These CAD systems are often misleadingly construed as true feature-based modelling systems” (Lim95).
- “In reality, features in these systems are merely viewed as macros that facilitate the creation, parameterisation and placement of specific geometry forms within a solid modeller” (Perng97b).

However crude the implementations are, many of them have made serious commitment to extend or implement feature-based modelling in the future. Commercial CAD systems, highly influenced by parametric or variational constraint-based technology (see section 4.4.2), have created some confusion concerning feature technology. In addition, some strange behaviours have been reported when editing feature-based models using some of these systems (Chen95).

Therefore, commercial feature-based CAD systems are considered not yet mature enough to be widely used as a basic resource for research on feature-related modelling problems.

1.4 THE VALIDATION PROBLEM

When a new methodology, technique or theory is developed to model the behaviour of a phenomenon it is necessary to validate the model. Validation is the process of checking that a representation satisfies the criteria established by the *domain characterisation* in target - the model. Conformance to the

criteria confirms the validity of the model representation (Rossignac90, Jablokow94).

Many authors have pointed out the existence of problems when using feature-based systems and the importance of the validation task (Faux86, Dixon87, Emmerik89, Shah90, Rossignac90, Shah91, Sreevalsan92, Requicha92, Pratt93, Duan93, Martino94a, Su94, Kim96, Kraker97) but few state what a valid representation is in terms of the feature technology. Therefore, even the origins of the validity problem are not completely clear.

Some of these problems related to representation validation have been referred to in the literature using the following keywords: manipulations, editability, operating, consistency verification, construction and changing feature-based models.

One question to be answered is which restrictions and/or verifications should be applied to a feature representation (preferably at the feature representation level) in order to guarantee that the model is within its domain and is really representing the artefact's *geometric semantics*. Defining a set of representation *validity conditions* establishes the criteria that must be applied to classify it as being in the domain of the model and thus a valid representation.

Although still lacking a proper clarification, architectures for the future feature-based design system have been proposed (Allada95, Kim96) where "feature validation" and "designer's intent" have been considered as necessary elements of such architectures. As a result, it can be seen that "feature validation" is an important and active research topic and thus needs to be further studied, which is one of the aims of this research.

1.5 DISTINCT VALIDATION ASPECTS

On analysing the validation problem, at least three aspects can be distinguished (Figure 1-4): Model Validation, Entity Validation and Model Representation Validation.

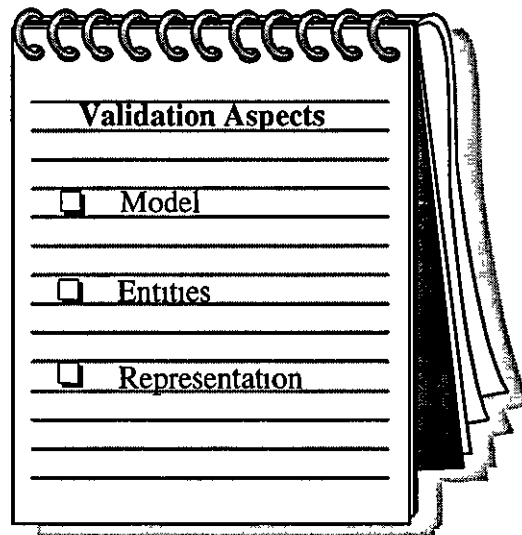


Figure 1-4: The Validation Aspects.

1.5.1 MODEL VALIDATION

Model validation seeks to prove that the model “does its job” in a variety of circumstances and that the model agrees with the “real thing”, at least to some extent. Most often, there are built-in restraints that apply to the model to guarantee the extent of the modelling and that models are within its representation domain. This is called *model representation validation*.

Feature-based modelling has already been accepted as a valid (and indeed, necessary) modelling framework (Dixon90, Denzel93) that will promote Concurrent Engineering (CE) (Lim95) and a better integration of CAD systems with other computer-aided engineering activities such as CAPP, CAM and CAE. This acceptance can also be inferred from the variety of feature applications that thrives in the literature (some of which are presented in section 2.3).

From the modelling perspective, feature-based models have the same modelling domain capability and limitations as the underlying geometric modeller. Therefore, a CSG-based system would not be validated as a good modelling framework for sculptured parts such as golf clubs or shoe lasts.

Rather, a feature modeller with an underlying “surface modeller” would be more appropriate for this domain (Mitchell96).

1.5.2 ENTITY ELICITATION

Once feature-based modelling is accepted as a valid alternative for a product domain, it is necessary to identify and validate features for that product domain. This gives rise to two activities: *entity elicitation* and *entity validation*. These two together are called the *elicitation process*.

The *entity elicitation* process is an identification methodology. Using manufacturing features as an example, this process, also called *featurization* and summarised in Shah and Mantylá (1995) and Mäntylä et al. (1996), is reproduced below:

- determine the scope of the product and processes to be covered;
- identify the individual process steps within the chosen scope;
- formalise the process steps as recurring process elements and identify process parameters and relationships between processes;
- identify recurring process sequences related to a certain type of geometry, and formalise the relation between the geometry and the process parameters of the steps of the sequence;
- call the resulting shapes “manufacturing features” and name each feature parameter.

It can be inferred that there are embedded *elicitation criteria* such as features being required to be associated with a (manufacturing) process step and that there are recurring entities.

Classification schemes have been proposed to ease the task of *featurization* and to facilitate the understanding of a feature domain and its functionality. This has been achieved by categorising features using shared behaviours and

characteristics. Various classifications have been proposed but it has been stated that “their differences emphasise the difference in feature views between researchers even when they share a similar interest in the same application (Mitchell96).

Figure 1-5 presents an example of a form feature classification scheme (Pratt85):

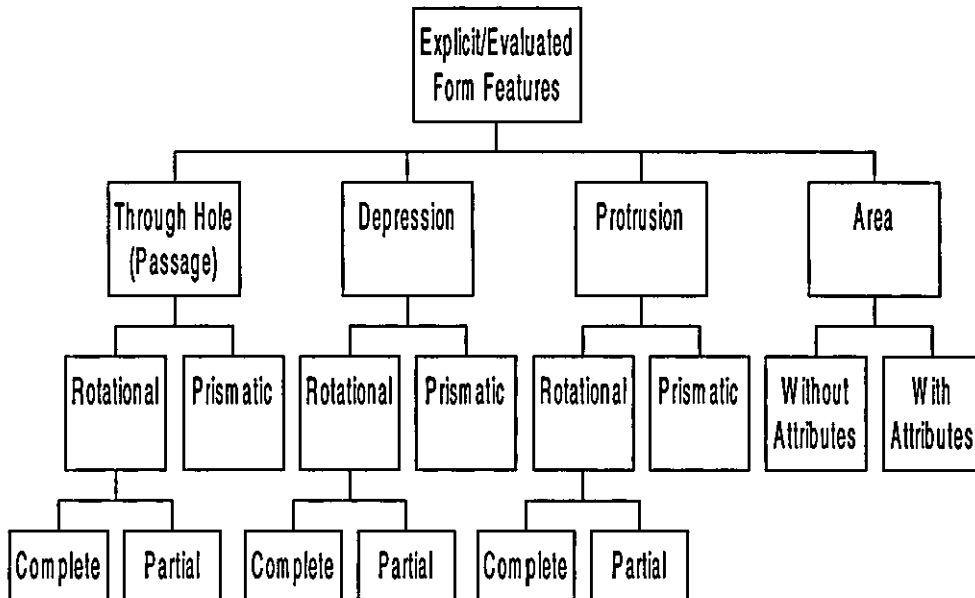


Figure 1-5: Pratt's Feature Classification.

The current thesis takes up an adaptation/simplification of Gindy's classification that is based on feature “external access directions” (EAD's, Gindy89). This adaptation is presented in Figure 1-6. Some of the features originated from this classification, accompanied by other details, are presented in Figure 8-19.

The subsequent step in the *elicitation process* is to better identify and enumerate individual entities (features) for use in the particular application context. This gives rise to taxonomies of entities (features).

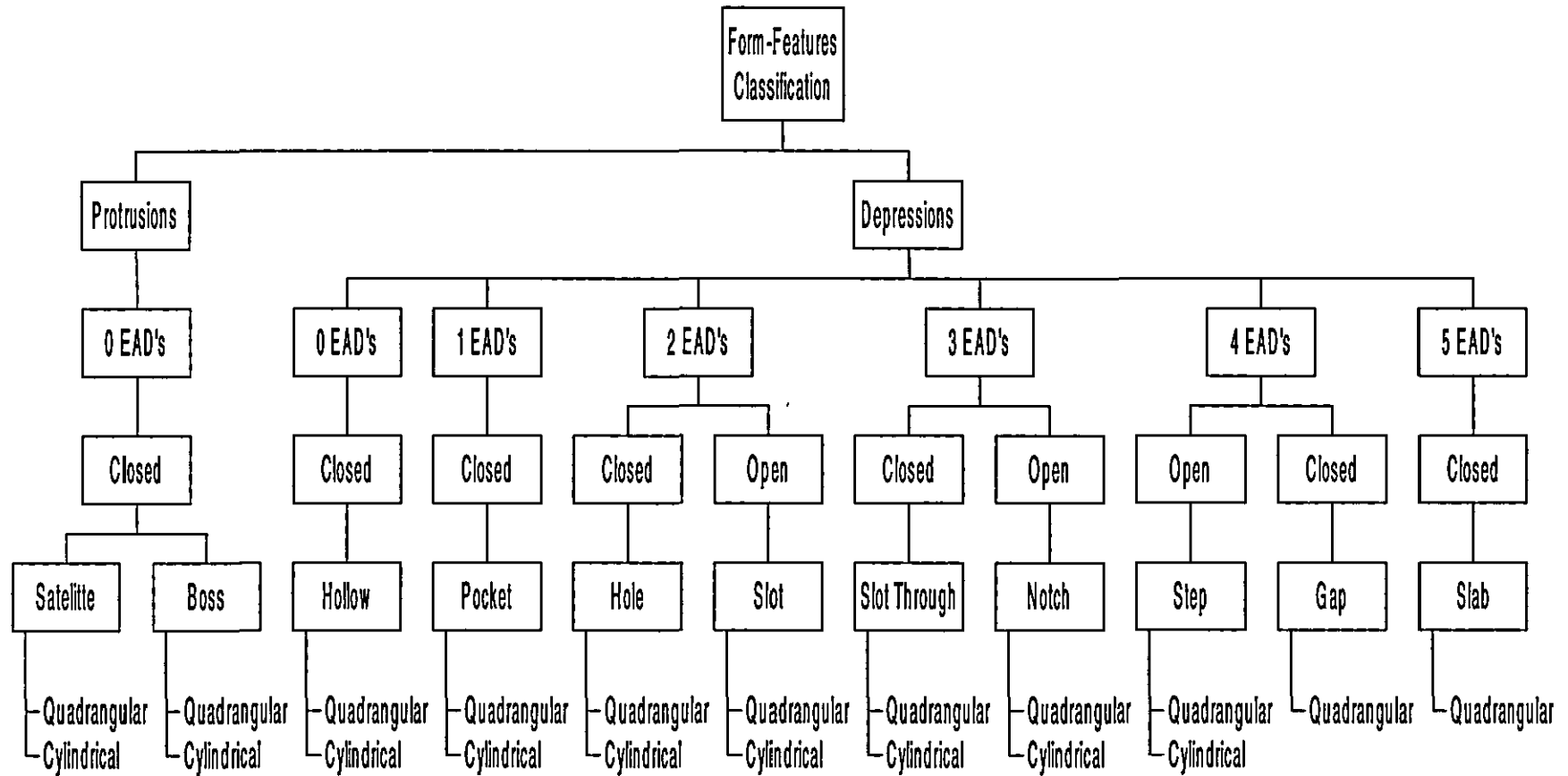


Figure 1-6: Adaptation of Gindy's Feature Classification.

Both Pratt's and Gindy's classifications produce features taxonomies from different views. As an example, Pratt's (Pratt85) feature taxonomy originated from his classification (Figure 1-5) and is reproduced in Figure 1-7.

Taxonomy "is the classification and naming of things ... in groups within a larger system, according to their similarities and differences" (Collins87). Therefore, it is considered that a taxonomy is the identification, naming and placement of entities in a (possibly already existing) classification that inserts specific elements into the classification.

Taxonomies have been categorised according to the pair of process-product type and/or their cross-section shapes (Pratt85): Rotational Features; Prismatic Features; Thin-Walled Features and others.

Examples of taxonomies can be found in Libardi86, Dixon87, Shah88c, Mäntylä89, Chovan91, Ovtcharova92, Kang93 and Rembold93.

However, there has not been wide agreement on the results of this entity *elicitation process*. One of the reasons relates to a current deficiency in feature taxonomies in that blending is absent from much of the research as it is seen as a "non-feature-related activity" (Allada95).

Blends are one of a few "sculptured features" common to predominantly prismatic parts, and yet are somehow considered separate and so not included as features (Denzel93, Mitchell96). Some exceptions to this include Laakko and Mäntylä's *transition* features (Laakko93), Chen's *modifying* features (Chen95) and Perng, Chen & Li's *fillet* and *arc* features (Perng90).

Another reason why the *featurization* result is not widely accepted might come from the fact that it has been considered that "any feature library (taxonomy) in any system can never be complete" (ElMaraghy93a).

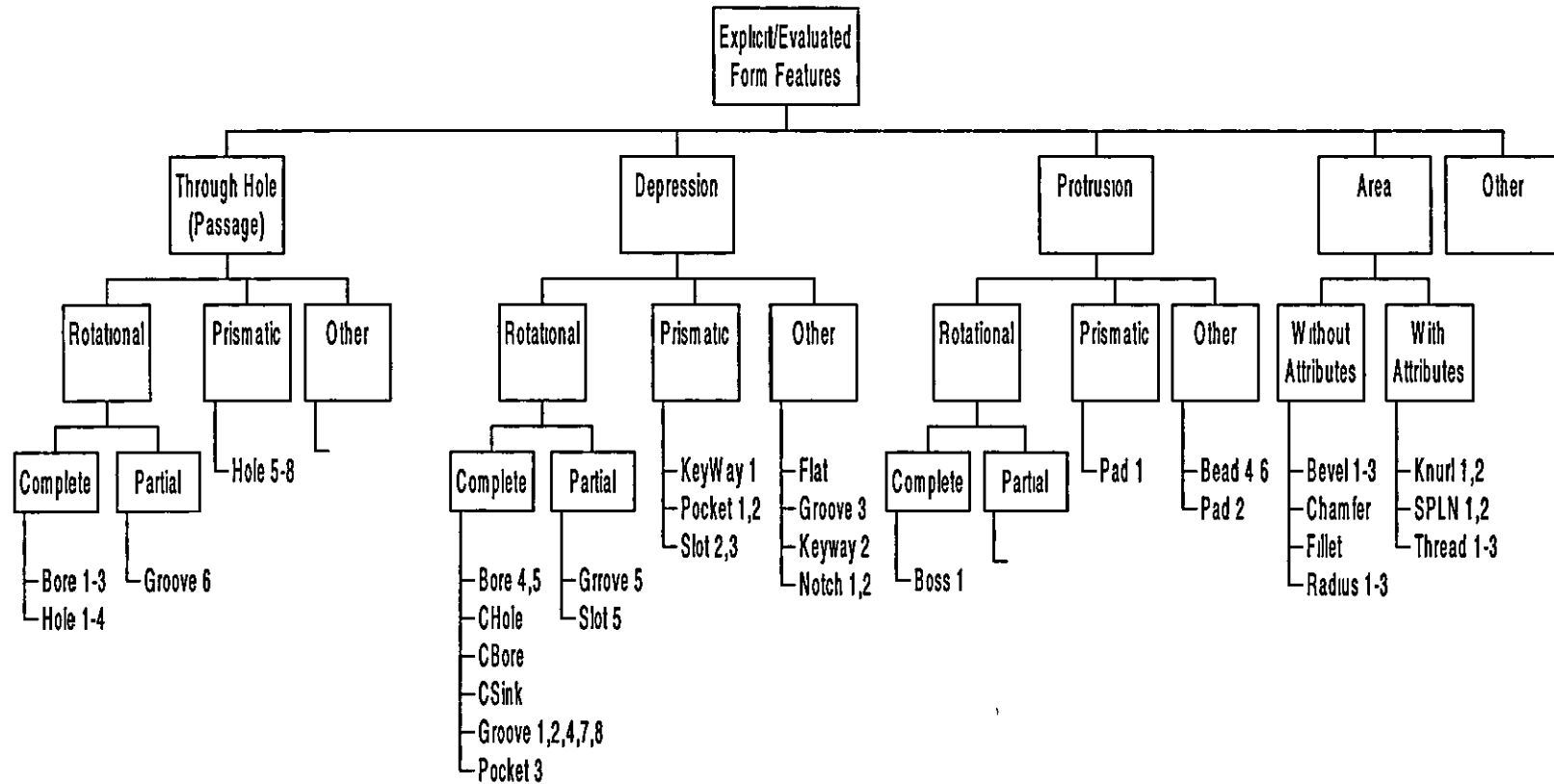


Figure 1-7: Pratt's Feature Taxonomy.

1.5.2.1 FEATURIZATION VALIDATION

After entity elicitation (*featurization*) a set of entity candidates (features) is produced and should be validated against the chosen product's *domain*.

Featurization validation represents the process of selecting a reasonably small (or minimum) subset of all feature candidates raised from the elicitation phase, in a specific domain, that demonstrates the best properties (including expressiveness and flexibility) to suit an application. Featurization validation thus requires a set of *validation criteria*. Examples of featurization *validating criteria* include (Mäntylä96, Shah95):

- *completeness*: is the identified set capable of creating all parts of the chosen domain?
- *unambiguity*: do the proposed parameters unambiguously identify a feature type?
- *simplicity*: are properties (and parameters) only included if they are of use in some application?
- *uniqueness or duplication*: can nearly identical features be united and the part still be uniquely modelled using the new feature?

It should be noted that the final set could be a sub-set of the elicited set depending on the *application*.

However, in the context of features the final validation process has been relegated to being considered of minor importance because:

- there is a close relationship between a feature's domain and its application, and therefore it is not easy to dissociate features from their application semantics;
- the feature classification process has been carried out in a way that emphasises the application needs (see dotted arrow in Figure 1-8 showing

the influence of an application over the definition of the classification, and therefore, over the elicitation process)

1.5.2.2 THE COMPLETE ELICITATION PROCESS

A formalisation of the *featurization* and *featurization validation* processes gives rise to a general *elicitation process*. The feature classification and taxonomy are important resulting products of this process, in addition to the final feature set.

The complete *elicitation process* is depicted in Figure 1-8. The following are the elements of this process that need to be specified in order to obtain the resulting set of entities (e.g. features, intents): *domain*, *elicitation criteria*, *classification*, *taxonomy*, *validation criteria* and *application(s)*.

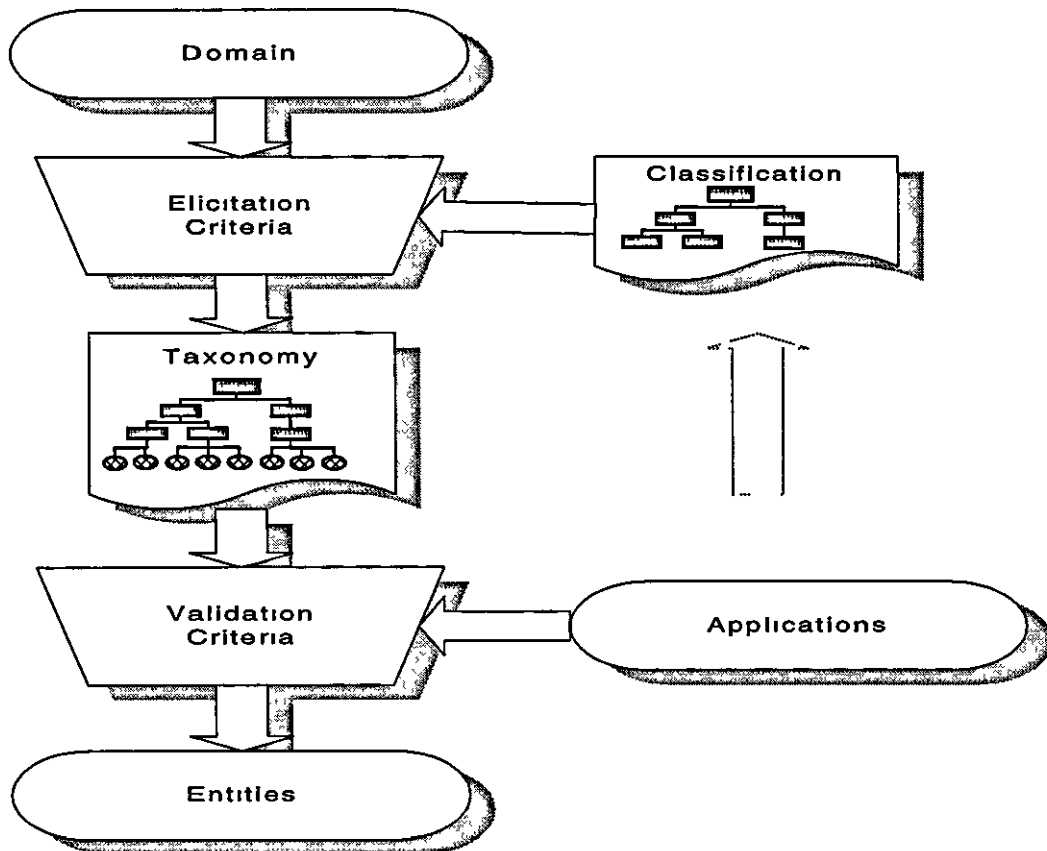


Figure 1-8: The Complete Elicitation Process.

1.5.3 REPRESENTATION VALIDATION

Having a library of features elicited and validated for the target domain, the designer can use them to model a part. The result is a model representation of the part in terms of the available features.

Operating feature-based model representations can easily produce invalid representations. An invalid feature-based model representation occurs when any of the behaviours, intentions or conditions pre-defined for any specific type of feature is violated.

An example of an invalid feature-based model representation is given in Figure 1-9. The component in the figure has been mistakenly modelled using a *through hole*, a *notch*, a *through slot* just beside the *notch* and a small *blind slot* feature.

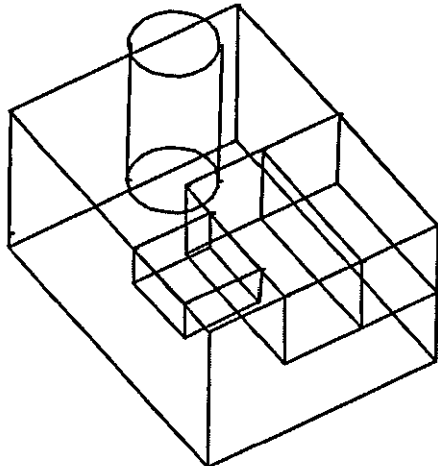


Figure 1-9: An Example of an Invalid Feature-based Model.

Various sources of invalidity can be attributed to Figure 1-9:

- a “dangling face” appears between the *notch* and the *through slot* features which causes the geometrical representation to be considered as invalid;
- a feature has been defined with the functionality of a *blind slot* while, in fact, it has the functionality of a *through slot*.

- the *through hole* has been defined with a possibly wrong parameter (height), and it is in fact a *blind hole* (in this text called pocket, see Figure 1-6 and Figure 8-19);

In addition, the fact that the *notch* and the *through slot* features were not defined as one “bigger” *notch* can be considered to be confusing

Even the most basic operation of adding a new feature to the part model can produce invalid situations (consider the model in Figure 1-9 before and after the addition of the through slot as the last operation). This happens because features, when placed in the representation of the part, can have their semantics changed. Some of the semantic changes that can happen include:

- *extension* by composing complex features from simple ones;
- *modification* when some properties are affected (e.g. length, width);
- *destruction* when a feature suffers the destructive influence of other feature(s).

To keep track of these semantic changes, to avoid or warn their occurrence and to try to correct them it is necessary to constantly verify the representation. This checking mechanism has to, at least, guarantee the correct use and meaning of the individual atomic features. However, as features are not isolated when applied in a design, their interactions also give rise to a set of design intentions that must be considered.

This verification process is called model *representation validation*. Thus, model *representation validation* is the process responsible for verifying the feature-based representation of a part to guarantee that atomic features are being used according to their assigned meanings and expected behaviours, and that the configuration of all those individual features within a single model is also meaningful to the extent of some criteria.

Feature-based modelling (FBM), and indeed Design-by-Features (DbF), systems are usually based on Geometric Solid Modelling (GSM) techniques.

However, one basic element that makes GSM so well established, important, popular and powerful, namely *Geometric Validation*, lacks a sibling in the FBM world. This is so because features add a layer of complex semantics to CAD systems which make it difficult to establish measuring means and are subjective to implement. The sibling of geometric validation in the FBM context is the feature-based model *representation validation*.

Feature-based representation validation is very important because it is the process responsible for guaranteeing the delivery of a valid representation (and therefore verified, useful and misrepresentation free) to downstream applications.

The model thus needs to be verified constantly. Situations need to be identified and dealt with, possibly by an automatic operation. However, features have no mathematical properties and their definitions are not widely accepted. Therefore, the behaviour of features and their role needs to be defined in order to obtain a validation system capable of analysing a feature-based representation.

Validating a feature-based representation is a very subjective and difficult problem to handle in the most general sense (Ohsuda89, Salomons93) and in fact depends heavily on the role the feature plays with respect to a particular application. It is a 'very difficult and obscure task because features themselves are not well understood with their extra meaning, purpose and objectives in addition to the embedded geometric data representation' (Rossignac90).

A new philosophy for defining features is required to help devise such a validation capability. This philosophy would define features by their functional intents at an objective, measurable and pragmatic level. This type of validation should not be confused with geometric or topological validations that are based on mathematical laws (Shah95). This distinction emphasises that it is possible to produce a valid *solid* model for it but it can still be invalid from the feature-based design perspective.

For instance, even if the dangling face had not been generated the in Figure 1-9, the remaining feature-based model would still be considered invalid.

1.5.4 EXAMPLES OF THE VALIDATION ASPECTS

Figure 1-10 presents the validation aspects applied to the GSM domain and the Boundary Representation (**B-rep**) method.

B-rep has been validated for modelling two-manifold polyhedral solids. Among other conditions for the solid, open shells, disconnected objects, dangling edges/faces, non-orientable faces, self-intersecting faces, infinite and nonsense objects are all disallowed. Needless to say, atomic geometric entities such as *points*, straight *edges* and planar *faces* in 3D Euclidean space are validated for this domain. Therefore, model and geometric entities are validated for the polyhedral solids domain.

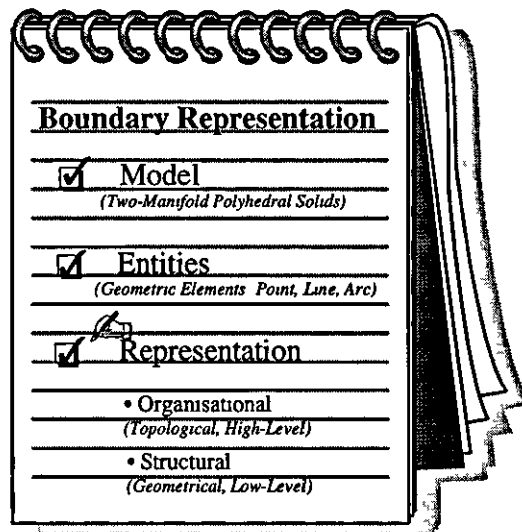


Figure 1-10: The Validation Aspects for B-rep.

B-rep representation validation conditions are divided into topological and geometrical sets that perform analysis at two different levels: the high organisational and combinatorial level and the low structural and metric level, respectively (Requicha80).

Examples of Topological Validity Conditions include: non-null pointers in the data structure, number of faces greater than or equal to 4, number of edges

greater than or equal to 6, number of vertices greater than or equal to 4, the relationship between the number of faces, vertices and edges conforming to the Euler-Poincaré formulae and each edge in a face belonging to exactly 2 faces of the model.

Examples of Geometric Validity Conditions include: each vertex must be a distinct point, distance between vertices must be greater than zero and all vertices in a planar face must satisfy the plane equation.

The elements shown in this example were reproduced from Jablokow94. Further details on B-rep representation validation can be found in Requicha80, Eastman84, Rossignac91, and Zeid91.

Feature-based modelling (the modelling technique) and features (the entities) are considered to have been validated. However, feature-based representation validation has no proper definition whatsoever and remedying this is a major objective of this research.

1.6 OBJECTIVES

1.6.1 MOTIVATION

This work has been motivated by a search for a more supporting feature-based CAD system. These systems in general allow the design of mechanical parts using an intermediate level vocabulary. They do not normally use high-level functional entities such as “conversion of rotational to translational motion” or “cooling holes”. Neither does the user need to worry about low-level representation aspects such as the placement of edges or points. In addition, although considered of higher level than current CAD vocabularies, feature-based modelling environments have not been widely used in conceptual design.

It is felt that feature-based CAD systems have sometimes used a vocabulary that addresses conceptual design and sometimes detailed geometric design

because of the lack of a complete vocabulary at the intermediate level of features.

This work seeks to establish through, a complete intermediate level vocabulary, a feature-based modelling system capable of performing validation of the representation as one of its supporting roles. Also, the methodology used in the search and in the validation process are of concern.

The careful selection of words throughout the text to describe some of the concepts defined in this research should be noted. For instance, the term *designer's intent* is sometimes used instead of the term *design intent* to emphasise the fact that *design intent* reflects the intention of the project or product while *designer's intent* reflects some of the ways that the designer uses to achieve the former.

1.6.2 OBJECTIVES STATEMENT

The objective of this research is to establish methods for assessing the correctness and integrity of feature-based model representations. It seeks to identify major elements that influence the validity of feature-based representations. These elements would enable the identification of an architecture for a feature-based validation system that would analyse, reason with and correct feature-based representations.

This approach would be able to better support the design task, raising awareness of many important aspects of feature-based design and guaranteeing the usefulness of the model for downstream applications, according to certain criteria.

1.6.3 SPECIFIC OBJECTIVES

- Understand and specify the validation problem and its various aspects in the context of Design-by-Features (DbF) systems.
- Identify major elements required to compose a framework for the validation analysis centred on the feature's concept.
- Specify and establish means to identify (and possibly correct if necessary) valid and invalid feature-based model representations.
- Define the roles that the “designer's intent”, “feature interaction” and “feature operations” entities/phenomena have in the context of validation analysis and devise a methodology to refine them for this task in a way that: (a) produces a complete vocabulary to aid assessment of the model's correctness/validity; (b) keeps them in an intermediate-level, the feature level; (c) maintains a DbF approach perspective of the validation analysis, and; (d) keeps their concepts and use separate from each other as far as possible.
- Investigate how much of this analysis can be performed at the feature class level as opposed to the feature type level, i.e. in an object-oriented implementation, how much of the analysis can be attributed to the feature class and therefore be inheritable by all feature types (objects).
- Determine means to integrate and organise the resulting vocabulary of entities in order to produce an architecture for a meaningful validation assessment.
- Test the feasibility of implementing such a validation framework in the form of a prototype system that illustrates some of the validation issues.

1.6.4 GENERIC SCOPE

This research concerns the use of a design-by-features approach (see section 2.6) to model individual mechanical parts. Orthogonal prismatic (laid parallel or perpendicular to the main axes) form features are of major concern, favouring analysis usually undertaken for manufacturing and process planning using milling and drilling processes. However, many of the ideas have considerable application potential in other areas (e.g. assembly).

This research has concentrated on three of the perceived outstanding research issues of Feature-based Modelling each of which has been recognised for some time but received inadequate attention:

- Feature Validation, as perceived by Salomons93;
- Feature Interaction, as perceived by Dixon90 and Allada95, and;
- Capturing Designer's Intents, perceived by Dixon90 and Salomons93.

1.7 RESULTING PUBLICATIONS

The following publications resulted from previous versions of the ideas generated throughout this period of research. They are listed chronologically and will not be referred to in the rest of this text.

1. *Hounsell, M. S. and Case, K. "Representation Validation in Feature-Based Modelling: A Framework for Design Correctness Analysis and Assurance". Proceedings of the 12th National Conference on Manufacturing Research (NCMR'96) (ISBN: 1 85790 031 6), Bath, UK, Vol. 1, pp. 256-260. September, 1996.*
2. *Hounsell, M. S. and Case, K. "Structured Multi-level Feature Interaction Identification". Proceedings of the 32nd MATADOR Conference (ISBN 0 333 71655 8), A. A. Kochhar (ed.), UMIST and Macmillan Press Ltd, Manchester, England, Vol. 1, pp. 495-500. July, 1997.*

3. Hounsell, M. S. and Case, K. "Intent-Driven Reasoning Priorities in a Feature-Based Validation System". *'Sustainable Technologies in Manufacturing Industries', Proceedings of the (IMC'97) 14th Conference of the Irish Manufacturing Committee, (ISBN: 1 897606 16 8), J. Monaghan and C. G. Lyons (eds.), Dublin, Ireland, Trinity College Dublin, Vol. 1, pp. 115-124. September, 1997.*
4. Hounsell, M. S. and Case, K. "Morphological and Volumetrical Feature-based Designer's Intents". *(NCMR'97) 13th National Conference on Manufacturing Research. Advances in Manufacturing Technology XI (ISBN: 1 9012 4811 9), D. K. Harrison (ed.), Glasgow, Scotland, Vol. 1, pp. 64-68. September, 1997.*
5. Hounsell, M. S. and Case, K. "Operating Invalid Feature-based Models". *(IDPT'98) Third World Conference on Integrated Design and Production Technology (ISSN 1090-9389), Editors: A. Ertas, D. Gibson, F. Belli, F. Veniali, R. Noorani and P. Chedmall. Berlin, Germany, Vol. 3, pp 151-158. July, 1998.*
6. Hounsell, M. S. and Case, K. "A Taxonomy of Feature-based Designer's Intents". *(IMC'98) 15th Conference of the Irish Manufacturing Committee, Belfast, Ireland, September, 1998.*

The first publication outlined the general principles of the feature-based representation validation that is discussed in chapter 3.

The second publication presented detailed analysis on one component required for the validation process that is discussed in chapter 5.

The third publication presented part of the priority organisation of the reasoning in the validation analysis (that is discussed in section 7.3).

The fourth publication presented some of the aspects being validated (that are discussed in section 3.5.2).

The fifth publication presented the types of remedy operations that help guarantee the validity of the model (that is discussed in section 6.3.3)

The sixth publication presented the whole taxonomy of feature-based designer's intent that has been obtained with this research (that is discussed in chapter 4).

1.8 STRUCTURE OF THE THESIS

This chapter presented the area of feature-based modelling and the problem of representation validation.

Chapter 2 presents a general literature review of important issues and particularly related work that utilises some sort of validation. However, some subsequent chapters also present reviews of work related to the concepts of their specific topic.

Chapter 3 presents a solution for the validation problem, the validation framework and its elements/entities, that will be discussed in detail in the three subsequent chapters.

Chapter 4 concentrates on the concept of Feature-based Designer's Intents, chapter 5 concentrates on classifying and identifying Feature-based Interaction cases while chapter 6 classifies Feature-based Operations.

Chapter 7 introduces the reasoning and organisation of the elements into a reasoning system.

Chapter 8 presents the implementation and exemplifies some of the reasonings.

Chapter 9 shows test parts that have been modelled elsewhere and how the prototype implementation deals with them.

Chapter 10 presents some findings and critically discusses the work.

Chapter 11 summarises the work, enumerates contributions and suggests future work.

1.9 SUMMARY

The various aspects involved in the validation of a modelling technique have been presented. It has been shown that in the case of feature-based modelling there seems to be no doubt that the modelling technique and its entities are valid components to express mechanical parts. However, little has been asserted concerning the validation of the model representation and its analysis.

It has also been shown that representation validation is an important and intrinsic aspect of feature-based modelling and that there is a lack of definition in the literature despite other types of validations being considered required, implied and applied. In addition, the role of geometric representation validation in a feature-based system has been established.

The objectives and the scope of this thesis have been laid down as well as the structure of the text.

The importance of the topic seems to come from the fact that validation is part of the "feature concept" (and therefore could be an inheritable property in an object-oriented approach) rather than part of the definition of each individual feature. Therefore, studying the validation problem could result in a clearer understanding of features themselves.

2. LITERATURE REVIEW

Feature-based technology is now a mature field (Case93a) and indeed, has already been incorporated into some commercial CAD systems. However, basic issues such as the properties and definitions of features still have an open interpretation despite the fact that much work has already been published on feature-based modelling. This chapter reviews some of these issues that are important in comprehending the context of feature-based validation and the following chapters.

2.1 FEATURE DEFINITIONS

Early detailed reports on various aspects of feature-based modelling technology implementation include Pratt85, Faux86 and Shah88c.

Good review papers on Feature-based Modelling include Shah91, Bronsvort93, Case93a, Salomons93 and Allada95.

Some analysis of open issues and suggested future research developments can be found in Shah90, Dixon90, Rosen93 and Mäntyla96.

Feature definitions presented exhaustively and in chronological order can be found in Shah88c and a definition classification in Bronsvort93. No single definition will be adopted. Instead, it is hoped that the feeling of what features are can be gathered from the following discussion and definitions.

Many authors have commented upon the variety of existing definitions and interpretations (Shah88c, Chung90a, Case93a, Lenau93, Pratt93) regarding the powerful and promising technology known as Feature-Based Modelling (FBM).

According to Sreevalsan and Shah (Sreevalsan92) the concept of features first appeared in manufacturing engineering in the mid 1970's when A. R. Grayer (1976, referred to in Shah91) was looking to automate NC part programming when it was felt that:

'Features represent shapes and technological attributes associated with manufacturing operations and tools'.

The need to automate the recognition of these features from a CAD geometric database gave rise to techniques that are now known as Feature Recognition (FeR).

Features were regarded as being exclusively geometry-driven and this has influenced many succeeding definitions:

'A feature is a region of interest on the surface of a part' (Pratt85).

'Features are defined as geometric and topological patterns of interest in a part model and which represent high level entities useful in part analysis' (Henderson90).

Some implementations establish direct relationships between features and manufacturing tasks (Grayer76, Choi84, Herbert90) whereas others are more flexible (see section 2.3).

Again, according to Sreevalsan and Shah (Sreevalsan92), the Design-by-Features (DbF) approach was first proposed and concepts to support form features with a solid modeller were first established by Pratt and Wilson (Pratt85) followed by Faux (Faux86). Thereafter, analyses of manufacturing heuristics were conducted to produce Feature Taxonomies (see section 1.5.2).

Features were then used to specify a part but were still limited to geometric/shape implications. Nevertheless, as CAD/CAM is not solely concerned with machining, but also encompasses other engineering activities such as conceptual design, features began to assume a wider meaning than simply geometric (see section 2.2) and the definitions started to change:

'A feature is a solid which can be manipulated (by Boolean operators like union, intersection and difference) over another one with defined validations' (Requicha92).

'Features are information sets that refer to aspects of form or other attributes of a part' (Lenau93).

Some definitions consider features that are related to various downstream applications such as mesh generation, finite-element analysis, turning, machining, assembly, etc. Hence, definitions began to incorporate such behaviours in a global sense in much the same way as the dictionary definition of features:

'A feature of something is a particular part of it (e.g. a component) or characteristic that it has, which you notice because it seems important or interesting' (Collins87).

Shah (1990) summarised the requirements for something to be a feature and is considered one of the fullest and most exact definitions (according to Ovtcharova94):

'A feature is a physical constituent of a part, is mappable to a generic shape, has engineering significance and has predictable properties'.

The search for a generalised definition has to a large extent failed because many authors came to realise that features are process-application dependent (Butterfield85, Cunningham88, Shah88b, Shah91, Pratt93) and this seems to be the only consensus regarding feature definitions. As summarised by Pratt (1993), 'features are expected to be used in diverse ways by organisations having widely differing product ranges, design methods, manufacturing

methods and facilities and general organisation philosophies'. Consequently, a single definition does not suffice.

It has been noticed though, that features could be implemented without any geometric representation (Dixon87, Shah88c) but they have, at least, geometric or shape semantics (Emmerik89, Ovtcharova92).

To summarise, *'The essence of the feature concept is that a product description not only says what the product is, but also contains implicit and explicit information on how it may be transformed to or from some other state'* (Case92a).

2.2 TYPES OF FEATURES

As can be seen from the variety of feature definitions presented in the previous section one can devise a type of feature and its taxonomy depending on what one sees as important or interesting for one's application. Different sets of features have to be defined in order to cater for different application areas or process-application pairs (e.g. "sand casting" - 'cost analysis", Denzel93). Therefore, a plethora of feature types can be found in the literature and include the following:

- a) **Functional** features only expresses the function and not the shape (Pratt85, Lenau93, ElMaraghy93b). They have also been called **Abstract** (Shah90) and **Design** features (Mill93). They are entities that cannot be physically realisable until all variables have been specified or derived from the model feature).

Functional features describe the part at an abstract level where there are several different possible geometries that could provide a specific solution (e.g. *bearing, sealing, ventilation openings, lubrication grooves, cooling slots, fixing holes, keyseats*). Despite the fact that abstract features could be incomplete at any given time, this does not prevent automatic reasonings being envisaged. Nevertheless, they must be a physical constituent of a part wherever information about them is complete (Shah90).

All sorts of features (especially form features, see item d below) have been called functional features (Zhang93, Martino94a) because other features are considered to have intrinsic functional meaning beyond simple geometry.

The following difference between functional features and form features (given by ElMaraghy93b in the context of CAPP systems) is accepted in this work: form features refer to recognisable shapes that can not be further decomposed, as otherwise they will reduce to meaningless geometric entities such as lines, points and surfaces. Form features may or may not have by themselves a functional purpose. Functional features are more natural for use by designers in comparison with geometric abstractions or form features.

- b) **Structural** features are non-geometric features that specify the relationships among form features. They have also been called 'embedded' features (Rimscha90) because they have no existence of their own without reference to their environment. **Assembly** features (see item j below) are examples of 'embedded' features. Although these embedded relationships are well understood and important, many authors have only considered parent-child hierarchies (see *connectivity* in Gindy93). Structural features can be said to have a similar meaning as the *skeletons* in Lenau93.
- c) **Physical** features (Kiryama91) provide the designer with knowledge about physical phenomena and mechanical elements at conceptual design stages. They consist of mechanical elements and "causal-related" physical phenomena that occur within the elements. For instance, a *wedge* has two intersecting faces and causes forces applied to a third face to act through the former two. Other examples of physical features include a *pair of gears*, a *spring* and a *pulley*.
- d) **Form or Geometric** (form) features are the most widespread kind of features used (and sometimes confused as being the only available features) in modern experimental and commercial CAD/CAM systems.

Form features have been considered the least application dependent type of feature because “they do not carry any specific non-geometric semantics” (Krause93). However, each form feature could have a set of possible manufacturing processes for obtaining the desired shape (a *hole* could be *drilled, bored or punched*). If such a strong geometrical and technological interrelation drives the vocabulary used to deal with form features, then they are called **Manufacturing** features.

‘Manufacturing features and processes mutually depend on, refer to and precondition each other’ (Vancza93) and basically consider material removal processes (Hummel89).

In addition, if form features are meant to represent shapes obtained by swept volumes of tool cutting paths they are sometimes called **Machining** features (Young93). Therefore they have been said to correspond to regions that can be cut by a machine tool (Requicha89b).

Alternatively, if the designer’s vocabulary or mechanical functions drive the terminology of form features, they can also be called **Design** features (Requicha89b, Rosen93).

Because it has been asserted that the shape of a part, and thus its form features, are results of the physical nature of the manufacturing process, form features have been subdivided according to process capability and a comprehensive sub-classification has been suggested (Butterfield85):

- **Prismatic** form features have been the subject of considerable effort in defining general taxonomies (Gindy89, Shah91, Ovtcharova92) and implementations (Anderson90, Duan93, Gao93) for shapes produced by extrusion, milling, drilling and similar processes;
- **Rotational** form features (Nielsen91, ElMaraghy91, Duan93) also called **Turning** form features are related to products with axial symmetry;

- **Sheet-Metal** form features (Cunningham88, Chung90a, Crawford93) refer to bending, forming and punching processes where the change in thickness is only incidental (Rembold93);
 - **Casting or Moulding** form features (Luby86, Cunningham88, Lee94) model investment casting, forging, injection moulding and similar processes.
 - **Sculptured** form features (Jones93, Mitchell96) model complex curved surfaces such as those found in golf clubs and shoe lasts.
- e) **Precision** features (Shah88a, Lenau93, Salomons93) contain explicit dimensions, dimensional constraints, surface finishes, and tolerances such as size, height, diameters, roundness, straightness, flatness and diameters.
- f) **Material** features (Shah88a) specify treatments to materials and surfaces, the material of the stock component to be used and its ability to produce specified physical characteristics such as rigidity, elasticity, durability and resistance.
- g) **Datum** features provide regions of the component from which the positions of the component on the machine table can be defined. They can be (Young93) *holes, corners and boxes*, or (Chen95) *points, axes and planes*.
- h) **Fixture** features (Young93, Pratt93) provide regions of a workpiece that can be used for fixturing. Fixture features include clamps, primary locations and secondary locations.
- i) **Technological** features (Shah88a) contain information about part performance and technological restrictions such as tool availability, machine operating variables (cutter velocity, feed velocity), directions of access, number of simultaneous operations and precision achieved by each machine;
- j) **Assembly** features are geometric relationships between (parts of) topological entities or features belonging to different sub parts of the whole

component (compounding what is called a 'handle lattice', Rimscha90). The main concerns are matching faces, accessibility and feasibility (Sodhi91, Molloy93, Harun96). A semantic sub-classification of assembly features can be found elsewhere (Ovtcharova92).

- k) **Manufacturability** features are process-capability dependent and are concerned with Mouldability (Lee94), Turnability, Machinability, etc.
- l) **Modifying** features (Chen95) or **blend** features (Laakko93, Denzel93) are localised geometric operations that alter the boundary configurations of parts. They represent sculptured features common in the predominantly prismatic domain and include *chamfers* and *fillets*.

The list of feature types seems almost endless and one can also find **analysis**, **tolerance** and **inspection** features (Sodhi91, Marefat93a, Pratt93a) as well as **production engineering** features (Vancza93, Mill93).

2.3 FEATURE-BASED APPLICATIONS

The variety of feature types reflects the wide variety of applications that use feature-based modelling. Some of the feature-based applications found in the literature include: Design for Manufacture and/or Assembly (**DFMA** - Shah88c, Jakiela89, Rimscha90, Ovtcharova92, Denzel93, Duan93, Harun96), Design-for-Mouldability (**DFMould** - Chung90a, Lee94), Computer Aided Process Planning (**CAPP** - Anderson90, Gupta92, Mill93, Young93, Vancza93, ElMaraghy93b), Automatic Inspection with Coordinate Measuring Machines (**CMM** - Requicha89a, Marefat93a, Medland93), Design-for-Fixturing (**DFFix** - Hayes89, Murray93), Setup Planning (Gindy93, Zhang94), Intelligent CAD systems (**ICAD** - Shah88a, Cunningham88, Ohsuda89, Nielsen91, Marghitu93), Automatic Group Technology code generation (**GT** - Srikantappa94) and Concurrent Engineering systems (**CE** - Fu94, Martino94a, Chen94, Lim95, Mäntylä96).

This variety of application also suggests that feature-based modelling is a very powerful technique that can be applied to a wide variety of engineering-related activities and this emphasises its importance.

2.4 FEATURE REPRESENTATIONS

2.4.1 B-REP, CSG, HYBRID SCHEMES AND ENHANCED REPRESENTATIONS

Features have been represented in four primary ways:

- by using one of the two major solid model representation schemes, i.e. Boundary representation (**B-rep**, e.g. those that produce an evaluated geometric representation) or Constructive Solid Geometry (**CSG**, e.g. those that produce a tree of unevaluated primitive volumes related via Boolean operators) schemes (Zeid91);
- by using a simplification of the B-rep or CSG. For instance, destructive solid geometry (**DSG**) has been used (Anderson90, Li90, Perng90, Waco94). This is a reduced version of CSG containing only the *difference* Boolean operator.
- by developing a hybrid of B-rep and CSG (it is understood that in such a hybrid scheme, primitives are represented as closed evaluated B-rep solids, these B-rep primitives are operated in a Boolean fashion and stored in a tree structure, ElMaraghy93b, Martino94a, Perng97b);
- by devising an enhancement to one of the previous approaches to accommodate feature-based information (Rossignac90, Stroud93, Su94, Mayer94).

Only a few attempts to model prismatic or rotational features were found not to be somehow related to the two major solid representation schemes (B-rep and CSG). For instance, “octree representation” has been mentioned (Tseng94, Allada95).

Hybrid B-rep/CSG implementation schemes seem to offer the best option for most requirements, with the minor disadvantage of redundancy and have been favoured by most research groups working with design-by-features systems (Pratt88, Shah90, Gomes91, Salomons93, Mill93, Denzel93, Suh95a, Allada95, Perng97a).

The advantages of such schema representation is related to the advantages of both B-rep and CSG representation schemes while also incorporating a beneficial bi-level parallel representation that is said to not only capture the history of the design (via a tree of simple set of operations) but also to offer detailed geometry if and when required.

It has already been predicted that a hybrid B-rep/CSG/Surface modelling approach will be used as the most generally applicable system of the future (Mitchell96).

2.4.2 VOLUME AND SURFACE FEATURES

Pratt classified feature implementations into **Volume** and **Surface** form features (Pratt88). Surface features are collections of faces of a part model that do not form a closed volume and are also a subset of the boundary of a volumetric feature - the solid (Requicha89b). Volume features are full-dimensional pointsets of the part or its complement that do identify a closed volume.

In a B-rep context, the essential difference between surface and volume feature representations is the existence in the latter case of *closure* faces which, when associated with the remaining feature faces (also called *support* face - Su94 - or *real* faces that actually lie on the part surface), define a closed and self-contained volume. To emphasise this difference in this context, an interesting, although dimensionally incorrect, equation has been suggested (Pratt88, Gomes91, Bronsvort93):

$$\text{"volume features} = \text{surface features} + \text{closure faces"}$$

Closure faces have become a persistent and beneficial aspect of most implementations. They help identify which faces of the generic feature template will have an impression on the part and which ones must not. Those faces that are absent in the explicit evaluated representation have also been called *imaginary* faces (Gindy89), *virtual* faces (Faux86, Silva90), *entrance* faces (Pratt88, Mayer94) or even *dummy* faces (Martino94b).

Although no particular disadvantage of using surface features can be emphasised, volumetric features have some noticeable advantages, especially in the context of design-by-features systems (Pratt88, Gomes91, Bronsvoot93):

- interaction between features can be easier to deal with;
- feature operations are simpler to implement. For instance, it has been asserted that volumetric features make delete operations easier and have other advantages over surface features (Pratt88);
- it is easy to extend feature concepts to general machining volumes;
- it is easier to manipulate and check the result of an automatic decomposition into delta volumes;
- the problem of individual faces belonging to different features is overcome;
- representation of more complex features composed of a number of simple features is simpler (Bronsvoot93).

2.5 FEATURE IMPLEMENTATION APPROACHES

On implementing feature-based modellers, Pratt classified the ways of defining features according to the status of the information as being **implicit** and **explicit** (Pratt85).

An implicit feature definition is an unevaluated one supplying the minimal amount of information to allow unambiguous evaluation when circumstances

require it. Implicit models produce very concise representations that resemble CSG models and similarly imply a procedural evaluation of the representation to obtain the exact and extensive information. Besides being compact, implicit feature implementations also use a parameterised representation of the feature volume at a very abstract level.

Implicit feature representation can be saved as a binary CSG tree where each node is a feature and where intersection Boolean operations are excluded. This binary tree can be efficiently traversed and manipulated (Su94, Mayer94, and Martino94a) in a sequential manner. Therefore, implicit features are also referred to as procedural or unevaluated features. Procedural models give rise to interesting problems that emphasise the non-commutative aspect of Boolean operations (Denzel93) but they are nevertheless easy to implement.

In contrast, explicit, evaluated or enumerated definitions refer to features that are sets of boundaries that together explicitly describe the actual status of the component boundary. Therefore, such representations are closer to a B-rep GSM core representation, which is extensive and complicated to manipulate from the feature's point-of-view.

Another interesting classification defined **intentional** or **geometric** features (Rossignac90, Tomiyama90) as an abstraction for accessing groups of geometric elements and for associating type and certain properties defined for all the features of a particular type. **Intentional** features are treated as hints and related to geometric elements through collections of unevaluated references. Some or all of these references are permitted to not correspond to any geometric element. On the other hand **Geometric** features are considered to be a collection of geometric elements that actually form a subset of the part's interior, boundary and/or complement.

2.6 SYSTEM APPROACHES

Feature-based systems can be divided into three main approaches from the interfacing point-of-view: Design-by Features (**DbF**); Feature Recognition

(FeR) and; Hybrid Design-by and Recognise Feature (HDR) approaches. Note the subtle difference in the arrows' directions in the respective figures (Figure 2-1, Figure 2-2 and Figure 2-3).

2.6.1 DESIGN-BY-FEATURES (DBF) APPROACH

The DbF approach (Figure 2-1) provides the user with a set of features intended to represent the designer's needs and a vocabulary for the type of component being modelled. Designers interactively select features, instantiate parameters and perform placements and positionings.

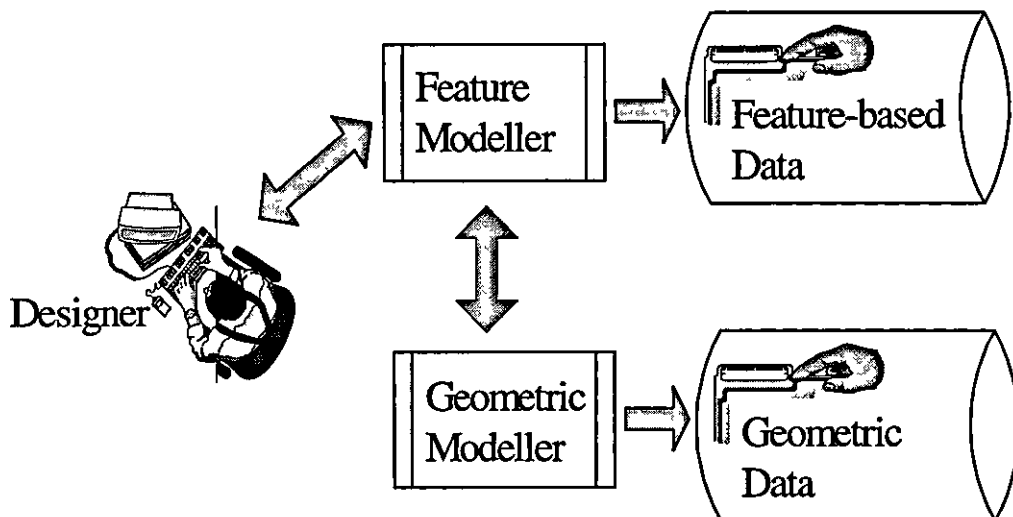


Figure 2-1: A Design-by-Features (DbF) System Approach.

A significant DbF advantage is that a great variety of non-geometric information can be stored and manipulated in addition to the geometry itself (Laakko93, Stroud93). A more natural design language, closer to the designer expertise, is used improving design's effectiveness and the set of features available helps towards standardisation.

DbF systems can ease the integration with design-related tools and downstream applications. It is considered that designer's intents, can be captured, manipulated and monitored. A more abstract, effective, conversational and iterative user interface can be built and integration with parametric, variational and constraint-based systems can be achieved.

A DbF approach disadvantage is that the designer is restrained to only a handful of already programmed features. Nevertheless, similar criticisms did not impede early CSG systems in becoming a major GSM representation technique.

It will be seen that direct manipulations of low-level geometric or topological entities has a drastic effect on feature models and is a complicated matter to cope with. Although feature-based systems imply an apparently simpler set of operations (such as *add* and *delete*) from which other operations can be built, features themselves have an intrinsic *union* or *difference* (but not *intersection*) Boolean behaviour, and therefore feature intersections produce another dramatic impact on feature semantics. Furthermore, (Boolean) operations are one of the centrepieces in the CSG representation scheme but there is no similar set of well-defined building operations in FBM. Therefore it can be concluded that the degree of flexibility and freedom found in conventional CAD systems is lost in DbF systems (Case93a).

DbF systems have been criticised in that if only features that correspond directly to manufacturing operations are made available by DbF systems designers would have to think in manufacturing terms even though they may find it unnatural to do so (Mäntylä96). In addition, a DbF system requires the user to become familiar with a new interface paradigm although this interface has been considered to be easier and more efficient.

2.6.2 FEATURE RECOGNITION (FeR) APPROACH

In a FeR approach designers interact through a conventional or GSM CAD system. After producing a complete description of the model, post-processing of the geometric data is performed to “discover” the intended features.

Advantages of the FeR approach include: a recognition process could be strongly optimised to a specific application (Laakko93); conventional CAD systems (and their well-known flexible, powerful and low-level manipulations) can be interfaced to other feature-based applications through FeR; there is

manipulation freedom and no need to invest in training on new interface paradigms; investment savings can also be expected if the FeR approach is used because existing conventional CAD systems will still be used and legacy files in traditional CAD formats can be saved and used as input to FeR and can also act as a converter to DbF systems.

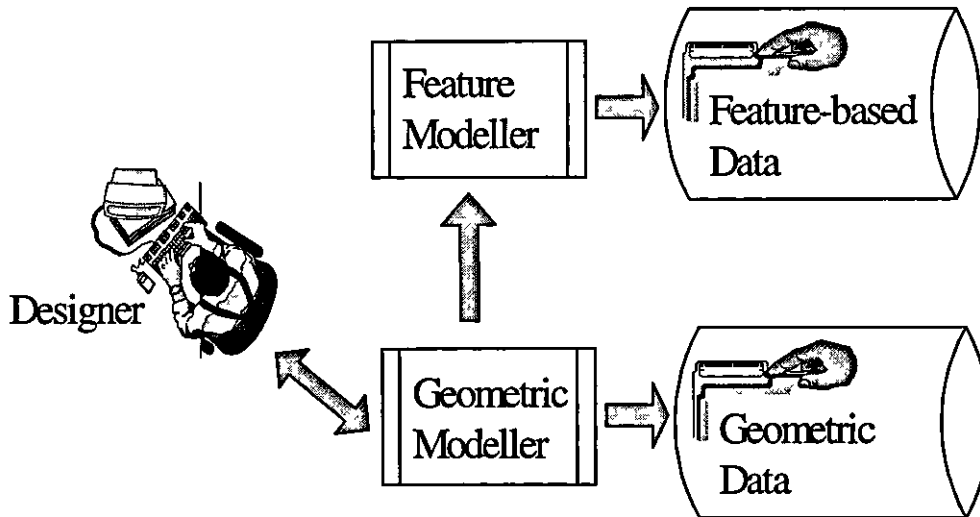


Figure 2-2: A Feature Recognition (FeR) System Approach.

However, FeR approaches have some remarkable drawbacks. They are usually hard-coded, very complex, time-consuming, difficult to achieve, lack generality (Mäntylä96) and sometimes are incomplete for the diversity of possible interactions among features (Gadh95a). Feature interactions make the recognition processes difficult and existing approaches only deal with interactions to a limited extent (Rosen93). FeR systems are also limited to the features that the procedures were prepared to recognise and if the number of recognisable features grows, the processing time grows combinatorially or exponentially (Gadh95a).

Despite much effort and significant improvements in FeR systems, various specialised features that capture special manufacturing processes cannot be “recognised” (Mäntylä96) and FeR procedures are not unique or standardised i.e. the same geometry may output different results for distinct implementations (Case93a); non-geometric information (such as tolerancing) can not be recognised and in some cases even some geometrical information can not be retrieved (Lenau93). Furthermore, FeR is a redundant process or, at least,

implies double translations (designer's intent to geometry then to CAD/CAM database) which makes it more prone to the introduction of errors.

The recognition is performed after the complete model is created, making it difficult to support concurrent designs or any other supporting analysis during ongoing designs and minor variations in the feature geometry/topology (such as *straight-slot* and *rounded-slot*) require a different pattern for searching and matching. Sometimes additional inferences are required to solve ambiguities (such as for distinguishing between a *boss* and a *slot* that have the same topology) which tends to make the procedures highly dependent on the underlying GSM representation scheme (for efficiency reasons) as well as application-dependent (Gomes91, Bronsvoot93, Gadh95a). Attempts at context-free feature recognition approaches have been recognised as suffering from both severe capability limits and performance problems (Mayer94).

2.6.3 HYBRID DESIGN-BY AND RECOGNISE FEATURES APPROACH (HDR)

In DbF, designers are limited to a number of already implemented features. This limitation is unlikely to be overcome because, although features are application specific and their interpretations are application dependent, the set of features used in design is large and sometimes even considered to be "not finite" (Shah91). Some attempts have tried to overcome this loss of flexibility (Requicha89b, Li90, Laakko93, Martino94a, Kim96) by providing ways of defining new features.

Nevertheless, a wide range of applications can cope with a limited number of features as they have been coping with other geometric limitations (e.g. in a Constructive Solid Geometry Modelling environment the designer is limited to a few pre-defined primitives). On the other hand such limitations could reflect standardisation and company practice that is sometimes very useful and required.

It seems hard to believe that FeR could "discover" high level features like structural and functional ones by any geometric reasoning.

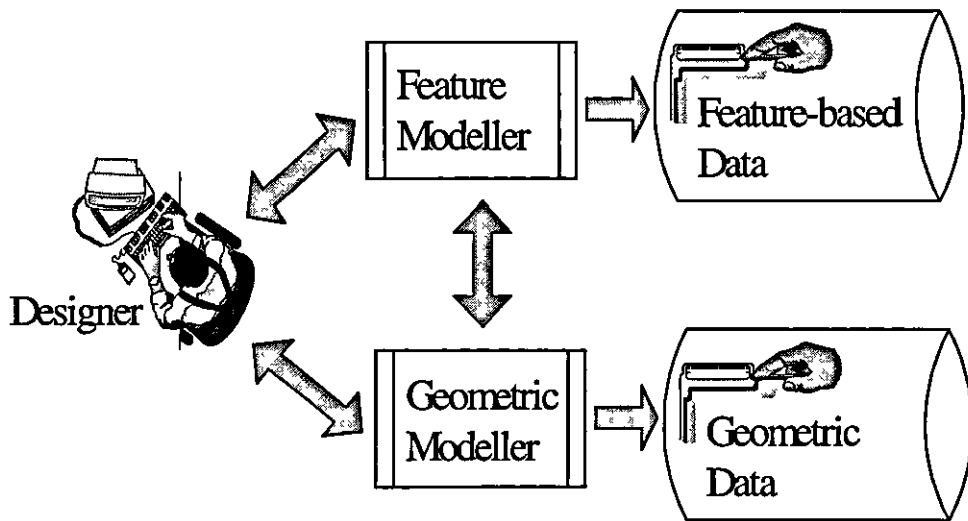


Figure 2-3: A Hybrid DbF/FeR (HDR) System Approach.

FeR is suitable for migrating to FBM for companies that only perform analysis and manufacturing of third party products and where data exchange is via some file standard. DbF systems are suitable for companies involved in all stages of the product life cycle and which are introducing new paradigms such as concurrent engineering. FeR allows the *interfacing* of CAD to CAPP, CAM and other activities while DbF allows their *integration* promoting DFM and DFA methodologies (Rosen93, Lenau93).

It can be said that features in DbF systems are more design-oriented while those from FeR are more application-oriented i.e. they are for different purposes, and having one set does not imply a lack of need for the other (Rosen93).

As both approaches have strengths and weakness some authors (Dixon90) have no doubt that the final solution to obtain an ideal feature based system is the integration of the DbF and the FeR approaches. Therefore, more recent thinking (Sreevalsan92, Zhang93, Laakko93, Martino94a, Lim95, Han97) has favoured a hybrid DbF/FeR (HDR) implementation as it has been recognised that DbF systems need some procedures usually available anyway in FeR approaches.

According to Martino94a, Sreevalsan's Master's Thesis pioneered the development of an Integrated DbF/FeR system followed by more complete

proposals from Laakko93, and Martino's group itself (Martino94a, Martino94b).

At least four ways have been identified to integrate FeR in a DbF-like system (Pratt93, Martino94a).

- as a FeR system itself, to convert legacy data from conventional and GSM CAD systems;
- to perform validation of operations;
- to solve some feature-interaction problems (Mill93, Suh95a);
- to convert features to application-specific feature-spaces (Bronsvooort93).

2.7 RELATED WORK ON REPRESENTATION VALIDATION

Although many systems have claimed to implement some sort of monitoring or validation system (see section 3.1), few have tackled the feature-based model *representation validation* problem specifically. Nevertheless, some points have been raised that are of importance to this work and are presented in the following discussion:

- a) Denzel93 pointed out how frequent and drastic is the difference between the feature before and after being incorporated in the model" (and even suggested keeping them as two separate classes). It was suggested that these situations should be avoided through "warnings" rather than trying to cater for them.
- b) Dixon and Cunningham (Dixon87) presented a Design-with-Features system where designer's intents are captured as constraints and argued the need for the system to have a monitor that ensures that the operations requested and performed by the designer are allowable and understandable by the system. It was also argued that this could represent a limitation or advantage depending on the completeness and sophistication of the

implementation and therefore, the value of the design-with-feature approach is very dependent on the monitoring and reasoning it can provide.

- c) Requicha and Vanderbrande (1989b) presented a set of four types of validation rules that surface features must satisfy. Requicha's rules include representation tests (presence and non-intrusion rules) as well as application-dependent ones (accessibility and dimensional rules) from a manufacturing viewpoint. They are performed at distinct levels: volumetric (non-intrusion and accessibility rules), surface (presence rules) and parametric level (dimensional rules).

It was argued that a good sophisticated architecture for a design-by-features system based on functional (design) features and CSG would imply a feature "*constructor*". A feature *constructor* would ensure that features are always valid. It would expand the feature operations into CSG, check the validation rules and automatically correct any violations. However, it was said that methods to implement such a system were not known.

It was exemplified that some validations have a local and global aspect. Local aspects can be tested throughout the ongoing design whilst global aspects can only be performed at, and therefore should be deferred to, a later stage (see local and global accessibility in Requicha89b).

- d) Work conducted by da Silva and colleagues presented an intermediate representation (lower than the feature level but higher than the solid representation level, Silva90, Srikantappa94) that is a language capable of expressing feature geometrical spatial relationships. These can be *interacting* (basically adjacent or touching cases) and *interfeature* relationships (when features do not physically interact but a spatial relationship exists). The language uses feature axes or faces as a reference to establish the *interfeature* relationship between each other and these can be planar, coplanar, offset, parallel, orthogonal, co-linear or angular.

Validation is achieved through rules that use these relationships to identify and operate changes in the model. The relationships are considered an

important and useful by-product of the feature extraction process (Srikantappa94) and are expressed in a semantic network (a graph).

It is believed that a representation of spatial relationships, such as the one proposed, allows the representation of manipulations and reasoning with the knowledge contained in mechanical parts and would allow this knowledge to be measured against ambiguity and completeness of form. Srikantappa and Crawford (1994) extended this approach to additive features and axi-symmetric features with the intention of automating Group Technology (GT) coding of feature-based parts.

- e) Sheu and Tin (1993) argued that capturing the designer's intent, parts functionality and geometry through a feature-based dimension-driven system would facilitate modifications of the model.

A feature representation scheme was presented where sizes and location dimensions are explicitly defined and 'constraints are defined to restrict the special behaviour of form features'. These constraints help prevent the violation of the validity of the part. A Feature-Dependency Graph (FDG) is part of the scheme and establishes the hierarchy of features and their dimensions. It was argued that (a) feature validation is context sensitive, (b) that complicated decisions can be made by the applications with the information supported by the FDG and (c) that constraining rules must be abundant to model the behaviour of form features and the requirements of applications.

- f) Case and colleagues (Gindy89, Case92b, Case93c, Gao93, Case94) have developed a DbF system called LUT-FBDS, Loughborough University of Technology Feature-based Design System. This system was fully integrated to a conventional B-rep solid modeller, Pafec Imager®, and aimed to define a feature representation scheme that has a widespread usage in CAD/CAM activities.

The B-rep representation influenced the feature taxonomy developed in the group (Gindy89), which emphasises "External Access Directions"

(EAD's). Relationships between features such as *tolerances*, *dimensions*, *parent-child* and *compound* features, are considered the most important and difficult task in the development of a DbF system. A hierarchical representation of the model is used and stresses the benefit and use of the parent-child relationship. This relationship is defined through sub-features (such as faces, edges and vertices) and geometric conditions (such as coincidence and containment of faces of different features).

The importance of the correctness of the representation for process and production planning, in particular after modelling operations was recognised. Of principal concern for a validation analysis was said to be geometric feature attributes that define position, orientation, dimensionality, class and feature relationships (such as parent-child, Case94). The proposed validation analysis aimed to detect and display all the possible changes in the attributes mentioned above when a feature is created or deleted (Case93c). The verifications performed are:

- Is a feature positioned or dragged beyond the boundary of the stock-material ?
- Is a feature intersecting another when it is positioned or dragged ?

The designer then has to decide whether or not accept any changes and to update the database. This analysis is performed via a set of rules for each feature primitive (Case93c) and for every manipulation case (such as "move in the +x direction" or "move in the -z direction"). Also, a mechanism that used the details of the B-rep feature description (such as the feature's EAD's and face properties) has been suggested to recognise class changes of a feature (Gindy89). These are ingenious, although lengthy, ways to analyse the model using the implicit data information, not always requiring access to the explicit (B-rep) data

- g) Stroud (1993) discussed various general classes of non-geometric information that can be associated with Boundary representation and the risks of this association becoming corrupted due to modelling

manipulations. Categories of usable information for describing a shape were enumerated:

- *basic shape* (edges, faces, etc.);
- *shape modifiers* (blends, screws threads, etc.);
- *features* (holes, slots, etc.);
- *attributes* (colour, price, origin, etc.);
- *constraints* on the shape (surface finish, tolerancing, etc.);
- *geometric frameworks* (centre lines, movement guides, etc.);
- *linkages* (static and movable assemblies).

Strategies were presented on how to keep correct associations between categories. The basic shape information (B-rep GSM model) was considered to be the primary and quantifiable model with which the extra information has to be associated. For features, a handling strategy suggested to maintain consistency is to have a set of modelling operations for each feature type.

These operations would then take care of preserving the integrity of the feature data structure and any associated information. However, it was concluded that further investigation was required to consider features as high-level information sets rather than collections of low-level elements, and to consider how they are allowed to interact. This analysis was left out of the discussion and low-level techniques were described instead. Localised feature recognition and extraction were suggested to handle information modification of modelling operations at *edges*, *vertices* and *point* levels.

- h) Shah and colleagues (Shah88c Shah88d, Shah88e, Shah90, Shah91, Shah95) proposed a system where the user has the freedom to define new *generic* features during a “setup phase”. During this phase an interpretative

language helps define rules for solid representation, user-defined parameters, parameter inheritance between hierarchically defined features, interpreting/mapping features for applications and also uses rules for performing validity checks on features.

Validity checking rules are called “cognition rules”. They represent dimensional constraints on how features can be used via size and placement restrictions. These rules must be defined for every single feature because it was considered that ‘there exists no universally applicable methods for checking the validity of features’ (Shah91). Therefore, a large number of rules have to be implemented to cover a wide range of features.

Four types of validation checks were identified (Shah95):

- *Attachment validation* involves the determination of the compatibility of adjacent features, compatibility of neighbours and compatibility of geometric entity type on which a feature is defined;
 - *Dimension limits* are restrictions on size parameters of a feature specified in order to maintain certain engineering meaning;
 - *Location limits* are restrictions on the position and orientation parameters of features;
 - *Feature interactions* (see chapter 5) are intersections of feature boundaries with those of other features such that either the shape or the semantics of a feature are altered from the standard or generic definition.
- i) Work conducted by Pratt and colleagues (Pratt85, Pratt88, Pratt93), recommended that the geometric validity of the feature-based model should be made dependent upon the transformations applied to the features in the model. Valid and invalid transformations are defined according to the compatibility between the transformation and the way features are embedded in the part.

It was considered that only valid transformations should be available to the designer, but every feature has different valid transformations according to its placement. A feature is only considered to be validly positioned with respect to a part body if certain rules are met. It was said that GSM validations are not enough to represent realistic engineering objects and therefore “another layer of validation becomes necessary”.

It was also considered that validation and revalidation processes are closely related to feature recognition and that there is a need for a standard method of describing features in terms of rules to which they conform (Pratt93). Boolean operations were used to check (validate) an unwanted topological change after model manipulation (Pratt88).

Pratt88 suggested that a reasonable first step towards a fully automated validation may be based on feature class rules. These rules include constraints on position, orientation, sizes and connectivity of feature faces.

- j) Zhang and colleagues (Zhang93, ElMaraghy93b) claimed to have considerably expanded Pratt’s basic idea to cover general cases.

It was pondered that the addition of a feature might invalidate the model if an interference occurred. Interference cases examined are “cover” and “collision”. Volumes and faces are used to check interferences (Zhang93, see section 5.3.2). Deletion was considered to cause a *wound* on the boundary model that was *healed* using a localised re-evaluation of the solid model.

Parent-child relationships were used with built in restraints (such as checking the compatibility between features) which served three purposes: (a) building a linkage between features; (b) inheriting the parent attributes, and (c) locating the child feature relative to its parent. Also, the hierarchical dependency between features and the use of an attribute-based language helped establish a parametric constraint-based environment. The system is claimed to be able to capture any production and manufacturing related functional data (ElMaraghy93b).

The feature-based modeller uses a hybrid CSG/B-rep data structure and was said to be capable of validating any construction of features by applying a few general rules by combining the advantages of solid modelling and feature modelling (ElMaraghy93b). Invalid models can be avoided by imposing manipulation constraints. However, a chain reaction problem originated by parametric relationships among different feature parameters was reported.

Because the same interference can be considered valid (intended) or invalid (inadvertent) according to the application, it was stated that it is necessary to consider the designer's intent in the validation method. Designer's intent is captured by simply attaching a special attribute to flag an intended interference or by making the system infer this respective connectivity between features and providing the user with a YES/NO option to validate the case.

The validation is performed whenever the product model is modified and invalid situations are left to the user to be corrected (Zhang93). Manipulations include the *addition* and *deletion* of features which are also used to implement manipulations others such as *edit* (divided in two steps: delete the old feature, and add the newly edited feature to the model) and *paste* (copy the feature parameters and add the new feature with different positioning parameters). *Add* and *delete* operations have separate analysis procedures within the validation method, and were applied to a hierarchical representation of the product. Their system deals with depression and protrusion features and both volumetric and surface features are evaluated in the validity check.

- k) Dohmen and colleagues (Dohmen94, Dohmen96, Kraker97) considered the problem of maintaining the validity (consistency) of a constraint-based feature-based modelling system that allows multiple views of the model. Constraints are used to specify feature validation rules and relations between feature instances. Each feature is considered to have a well-defined meaning expressed by constraints describing feature validity

conditions (Kraker97). Maintaining a feature's meaning, i.e. constraints, is called feature validation.

For a single view, validation constraints are divided into:

- *shape* constraints, which correspond to the type of feature shape, e.g. a block for a *slot*;
- *attach* constraints, which specify how a feature element (e.g. a face) contacts and aligns with an existing feature in the model. Attachments are used instead of parent-child relationships;
- *semantic* constraints, which specify topological properties of feature elements such as which element must or must not lie on the product boundary;
- *geometric* constraints, which specify geometric relations such as *parallelism* and *distance* between feature elements;
- *dimension* constraints, which specify intervals for the value of feature parameters;
- *algebraic* constraints, which specify equations constraining feature parameters.

A Constraint Manager applies several dedicated constraint solvers and deals with all types of constraints adopting the following solving sequence: *attach*, *shape*, *fix* (automatic constraint that specifies that the value of its variable may not be changed by the Constraint Manager), *geometric*, *dimension*, *algebraic* and finally *semantic* constraints are solved.

- 1) In Rossignac90, *validity checks* were said to assess the compliance of the feature-based models with the designer's intent. It was shown how a rich geometric representation scheme can be used to simplify the expression and evaluation of validity rules. It was suggested that, because features can be invalidated by subsequent creation of other features, in order to assess the

validity of the design the *intentions* of creating features must be preserved and methods for accessing the corresponding geometric elements and testing the compliance of these elements with feature validity rules should be available.

An extended mixed-dimensional boundary representation scheme was proposed to represent a solid part and its additive and subtractive volume features. *Intentional* features and *geometric* features were presented. *Geometric* features are the evaluated geometric embodiment of the feature while the *intentional* features are unevaluated abstractions for accessing groups of geometric elements and for associating with them a type and consequently certain properties. This association indicates a designer's intention to have that specific feature type in the model. However, *intentional features* are not *implicit* features (see section 2.5) because they are considered only as hints and could have references that do not correspond to any geometric element of the model's boundary at some particular stage of the design process.

It was assumed that no automated solution exists to correct the side effects of editing operations and that human intervention is necessary. To help the designer validate the model facilities for interrogating important properties are suggested. These properties are called *validity*.

The scheme improved the performance of updating the B-rep of a part model when a volume feature is modified. It was asserted that the validity criteria are domain dependent and two validation levels were considered important.

- The first, *individual* level, represents the verification that an *intentional* feature has associated to it a geometry that satisfies explicitly the requirements for that particular type.
- The second, is the *relational* level, where the relation between several features is needed to assess the validity of complex parts.

m) Su and colleagues (Su94, Mayer94) presented the Extended CSG Tree of Features (ECTOF). Features are tree nodes that comprise the explicit representation (*self-contained* or *basic volume*) as polyhedral winged-edge B-rep data structures and the implicit representation (parameters and other data to model the shape).

Representation validation was performed to maintain the *last* user's intent (Su94), regardless of possible intersection with other features and regardless of the feature's respective position (level) in the tree. To achieve this algorithms to rearrange the features in the tree were presented. Only simple orthogonal interference cases that generate *basic volumes* or empty sets were examined.

Rigid transformations (such as translation and rotation) and parameter editing are allowed as well as manipulations of feature size and shape as long as it remains a *basic volume*, which is a simple, closed set in a 3D Euclidean space bounded by a finite number of hyperplanes (Mayer94).

Besides rearranging the tree, reasonings were presented to decompose intersecting features and for removing redundant features. A three-phase sequential analysis and resolution of the interaction problem was presented.

- the first phase identifies "simple interference" cases that can be reasoned with. The interference may require the ECTOF to be rearranged in order to be consistent with the user's intention. Next, interacting features are split to identify effective volumes (and therefore, effective features) and to remove obsolete parts.
- the second phase reclassifies the remaining features to their correct types. This phase was designed to be activated by the user.
- the third phase groups and flags unresolved intersecting features as complex (and thereafter considered "resolved complex feature sets", Su94).

Validation is, though, a method to solve the feature interaction problem in the proposed representation scheme.

User's intent was considered to be the last operation (insertion or manipulation) although no option seems to be given to the user regarding splitting and removing interacting features and reclassification must be requested explicitly. Also, the design history (usually saved as the hierarchy of nodes in the CSG-tree itself) is lost.

n) Martino, Ovtcharova and associates (Martino94a Martino94b, Ovtcharova94) studied the integration of a DbF and a FeR system. It was argued that a DbF-like modelling environment with a FeR-like mechanism seems to be the solution to an efficient feature-based modelling system in a concurrent engineering scenario. The FeR mechanism can be used in three different ways:

- as a standard recognition approach from geometric models;
- as a mapping mechanism to taxonomies of application-specific features;
- the recognition process can be also responsible for maintaining the feature-based model consistency when degenerations or interactions with another feature makes the feature lose its characteristics.

The last item was called feature validation and was done by a localised feature recognition process (Martino94a). It was suggested that some interaction cases are better left unresolved depending on the application context.

The mechanisms used to update the explicit evaluated feature-based representation are *simplification* (merge features, reducing the complexity of the representation, Martino94b) and *subdivision* (splitting features, dual operation of *simplification*). These operations are responsible for producing alternative representations in different application contexts.

Combinations of predefined features can be defined by the user as new features. However, new features defined by means of the solid modeller can only be completely defined via programming due to the complexity of the new shape, manipulation tools, internal representation and recognition procedures. An intermediate representation was suggested to bridge geometric and feature-based models. The geometric model is regarded as the link between all feature-based models and the collection of all feature-based models creates the product model.

The user can interact via a feature modeller or the solid modeller. However, to reduce violations and degeneration when manipulating the boundary elements of the design, feature parameter constraints are used to restrict manipulations (Ovtharova94a) and no complex manipulations are allowed at this level (boundary elements can not be added explicitly or have their attributes changed - such as changing a straight *edge* into an *arc*).

- o) Kim and O'Grady (Kim96) proposed a validation formalism for the design process based on features. Four model validation levels exist and are used to characterise feature operators:
- *Syntax* level. Verifies that the model P is described only with the vocabularies of the feature algebra (which is part of the formalism and accounts for a feature taxonomy and two feature operators).
 - *Domain* level. Verifies that the model P is valid at the *syntax* level and that P satisfies a set of [geometric] domain integrity rules, e.g. the solid model for a mechanical part should be two manifold.
 - *Feature* level. Verifies that the model P is valid at the *syntax* level and all features assigned to the model maintain their semantic meaning despite feature interactions. This was said to be dependent on the feature definition and on how the design was performed.

- *Product* level. Verifies that the model P is valid at both the *domain* level and the *feature* level and the model P can be mapped into a realisable set of attribute values (defined via “concurrent engineering” constraints).

The formalism included a sequential loop-based algorithm responsible for maintaining the model’s validity. However, the *model* validation scheme performed by feature operators only covered a portion of the validity of the model and *functional* validity checking was also necessary (through the verification of function-to-form transformations which are the mappings of functional requirements onto features).

- p) Bidarra and associates (Bidarra93, Bidarra94, Bidarra96) presented an approach to validating a feature-based model that is “an intent to encapsulate interaction detection and reaction methods in each feature class definition, thus providing an automated mechanism for feature validity maintenance throughout the interaction phenomena” (Bidarra93).

Features are expected to exhibit a specified behaviour for a respective feature class. These are predictable properties, associated with some definite engineering semantics, expressed in terms of the feature associated volume (FAV), local morphology (the additive or subtractive nature of a feature’s volume), the characteristics of the feature associated boundary (FAB, subsets of the feature’s boundary that do or do not actually belong to the model’s boundary) and “a high-level graph representation of interactions, called FIG. FAB is divided in *semantic entity* (SE) sets that specify and individualise the behaviour for a respective feature class (these are either positive SE’s - boundaries that are present in the final model - or, negative SE’s - feature boundaries that are absent in the final model). The essential subset of the FAB for a given feature class is called a definitional entity set of that feature e.g. a *slot* feature should always have the definitional entities *roof* and *floor*.

Interaction cases identified are (see section 5.3.2 for details):

- *topological* interaction, an overlapping interaction that maintains both feature parameters and a semantically complete definitional entity set;
- *transmutational* interaction, that causes a given feature to exhibit a definitional entity set of another feature class;
- *geometric* interaction, that causes some dimension parameters to lose their correspondence to the actual feature geometry;
- *closure* or *absorption* interactions, closed feature boundaries that become open and vice-versa.

In Bidarra96 the following interactions were added: *splitting*, *disconnection*, *clearance* and *general*. These and the previous ones represent a classification of the interaction phenomena by their functional (geometrical or topological) or technological meaning. *Semantic constraints* were said to be the key to specifying validity conditions. *Semantic constraints* are predicate expressions that establish the feature's *canonical status* which is the definition of the semantics (positive or negative) of every boundary in the FAB. By these means, Bidarra's scheme is able to monitor every operation such as *insertion*, *removal* and *modification* within the DbF system and recognise the feature's class producing the feature's valid (complete) or invalid (intentional) status.

Constraint-based validation, where operations that invalidate the model are rejected or forbidden, were considered too rigid. Instead, ideally, the system was said to automatically adapt the model to get a valid one, although the user should be consulted (Bidarra96). Also, it was stated that most validity violations are caused by feature interactions which arise from modelling operations.

- q) Perng and Chang (Perng90, Perng97a, Perng97b) discussed the dynamic editing problem of a Design-by-Features system. In editing a feature-based part two problems for the part description were encountered: (a) changes in

the boundary and (b) changes in the destructive solid geometry (DSG) representation. Seven orthogonally prismatic (parallel or perpendicular to the coordinate axes) volumetric machining features were represented as DSG-nodes (a volume and a difference Boolean operator) and the equivalent B-rep evaluation. Volumetric feature interaction properties (cases) were detailed and used to derive localised modification functions.

The dynamic editing manipulations are *adding*, *editing*, *modifying*, *stretching* and *shrinking* a feature. The validity of the model was guaranteed via restrictions on the manipulations such as:

- not allowing a change in the feature's type;
- constraining the positioning reference of the feature to lie on the surface of the existing model;
- preventing the volume of the feature from exceeding the volume of the existing model;
- not allowing a modified feature to be enclosed within any existing feature.

If a valid manipulation is used, a valid representation (CSG tree and B-rep evaluation) can be computed efficiently according to the interaction properties (cases). It was found that the cases of enclosure and intersection dictated the way the representation is updated. According to this, as a first stage, cases can be identified where an efficient local update can be applied to the B-Rep (stage 1) without the need to completely re-evaluate it from the DSG tree. A second stage applies similar reasoning to update the DSG-tree. Without complex effort in defining an augmented/modified solid representation scheme, the feature redundancy of a part description is examined by spatial enclosure checking.

2.8 SUMMARY

The variety of definitions, types, taxonomies and applications of features emphasise two important aspects: features are well accepted as an important development medium for CAD systems, but little agreement exists on their formal characteristics (implementation, role as a 3D modelling environment, etc.). Possibly this is because of the very fact that they have been applied to such a plethora of applications.

In whatever context features are considered, they are the carriers of information beyond simply geometry and topology, but nevertheless the information carried is closely related to geometry and topology.

It has been shown what features are, methods of implementing features using various solid modelling techniques and how to approach this implementation. These techniques give rise to various issues in validating feature-based model representations. However most of the validation approaches are related to how features have been implemented rather than validating the concept of features.

The next chapter discusses the various validation approaches in the literature and introduces a validation methodology centred on the feature's expected behaviour which is closely related to the concept regarded as features.

3. FEATURE-BASED VALIDATION

Feature-based modelling allows extra meaning to be added to geometry, but lacks the equivalent representation verification formalism that exists in conventional and Geometric Solid Modelling (GSM) computer aided design (CAD) systems. A framework for a Design-by-Features (DbF) system with representation validation is presented that supports intent-driven modelling, encompasses existing low-level geometric verifications and incorporates operations to assure its correctness.

3.1 FACETS OF VALIDATION

Different feature-based modelling implementations have interpreted feature-based representation validation differently and were found to perform one or more of the validations described in the following sections.

3.1.1 GSM VALIDATION

Systems that use well-known solid representation techniques (such as CSG, B-rep or Hybrids) have implemented feature-based representation validation as GSM validation (Dixon87, Requicha89b Kang93, Duan93, Bidarra94, Perng97a, Perng97b). Possibly this is so because, although feature data

structures are separate from the basic shape of the part, they contain basically the same sort of information (Stroud93).

For these systems, feature-based validation has been considered to be the verification of the existence of a “proper” valid evaluated GSM counterpart of the implicit unevaluated form of the feature representation.

3.1.2 GSM-LIKE VALIDATION.

Feature based modellers supported by an augmented or modified geometric solid representation scheme usually consider feature based validation as a GSM-like validation (Rossignac90, Gomes91) in the sense that their data structure integrity analyses are similar to those found in GSM representation schemes, but using a modified representation scheme arrangement. This is related to the effects of the feature manipulations on the representation scheme and the ability of the scheme to represent features.

3.1.3 CONSTRAINT VALIDATION

Systems that implement features on top of geometric constraint-based systems consider that feature-based validation has to be done by resolving conflicting parameter relationships that, for these systems, are associated with or are part of the feature (Nielsen91, Emmerik91, Sheu93, Bronsvoort93, Ovtcharova94, Shah94b, Dohmen96, Kraker97).

3.1.4 MANIPULATION RESTRICTIONS

Some systems restrict the availability of manipulations that are more likely to produce complex, faulty or unknown situations from the system’s implementation point of view (Pratt85, Dixon87, Stroud93, Denzel93, Shah94b, Mäntylä94, Martino94b, Su94, Mayer94, Zhang93, Perng97a).

For instance, “cognition rules” have been defined (Shah88e) as size and placement constraints that are evaluated to ensure the valid use of features.

However, different features in different situations can have different sets of allowed or disallowed manipulations and thus identifying those allowed manipulations could become a complex task in itself. A more linear and simplified alternative is to limit the manipulations to a small number that can easily be handled.

3.1.5 ASSOCIATIVE VALIDATION

Some systems that claim to integrate DbF and FeR, and where low-level topological elements can be manipulated, implement validation by verifying that all low level geometric entities are associated with some feature (Sreevalsan92, Laakko93, Martino94a).

3.1.6 INTERSECTION VALIDATION

Others systems interpret the validation problem as the solution of problems originating from (geometrically) overlapping intersecting features only (Case93c, Salomons93, Su94, Mayer94, Suh95b, Perng97b).

3.1.7 RULE-BASED VALIDATION

Some systems present the validation problem in a broad sense, including considerations and rules that express how features should behave. These rules state behaviours such as the compatibility of a feature's neighbourhood, size and positioning (Shah95), aspects of manufacturability (Silva90, Gadh95b) or accessibility (Requicha89b, Rossignac90).

3.1.8 CONSISTENCY AMONG REPRESENTATIONS

Some systems that allow multiple representations of the object in different feature representation spaces (to represent it more appropriately for different audiences and to minimise the "*ribs versus slot*" or "designer versus engineer's view" problems) and that allow manipulations on the models in any of these views (promoting concurrent engineering) consider validation as the

maintenance of consistency among these representations (Shah90, Bronsvoot93, Dohmen96, Kraker97).

3.1.9 FEATURE'S TOPOLOGICAL VALIDATION

Some systems perform validation of a feature model through inferring geometric properties such as concavity/convexity edges/loops and verifying the consistency of the representation with topological rules established for a specific feature-type (Rossignac90, Stroud93, Zhang93, Duan93, Bidarra94).

3.1.10 VALIDATION VIA RECOGNITION

Some systems perform global or local feature recognition, especially when an intersection is detected, on the GSM evaluated data to compare or update a (parallel) feature-based implicit model (Stroud93, Pratt93, Martino94a).

3.2 REPRESENTATION VALIDATION

3.2.1 TWO LEVEL VALIDATION

B-rep GSM representation systems were shown (see section 1.2.2.4) to have two levels of validity conditions: organisational (topological) and structural (geometrical). Similarly, Krause93 has categorised some of the fundamental challenges in product modelling in terms of syntactical and semantical consistency.

Existing feature-based validation systems are basically geometry-driven. However, it has already been suggested that 'another layer of validation becomes necessary' beyond the geometric one (Pratt93). Low-level information modelling and integrity handling has been considered transparent and therefore poses no great challenge for the product modeller, at least those based on B-rep (Stroud93).

It has been affirmed that modelling systems that fail to notify (or acknowledge) a change in a feature's functional meaning are in essence only geometric modelling systems, and not real feature modelling systems, because they do not maintain the meaning of features (Bidarra96).

Low-level integrity handling (for all GSM techniques that have been applied to feature-based modelling) poses no problem because of the extensive research already devoted to this topic. Therefore, a validation approach centred on the feature's concept and implied designer's intents is needed (Sheu93, Dohmen94).

It is also considered here that feature-based representation validation is divided into two levels (see Figure 3-1):

- Considering that the majority of feature modellers are integrated in some way to an underlying GSM system, it can be said that both sets of GSM validity tests - geometrical and topological - represent the structural or syntactic validity conditions for feature-based representation validation.

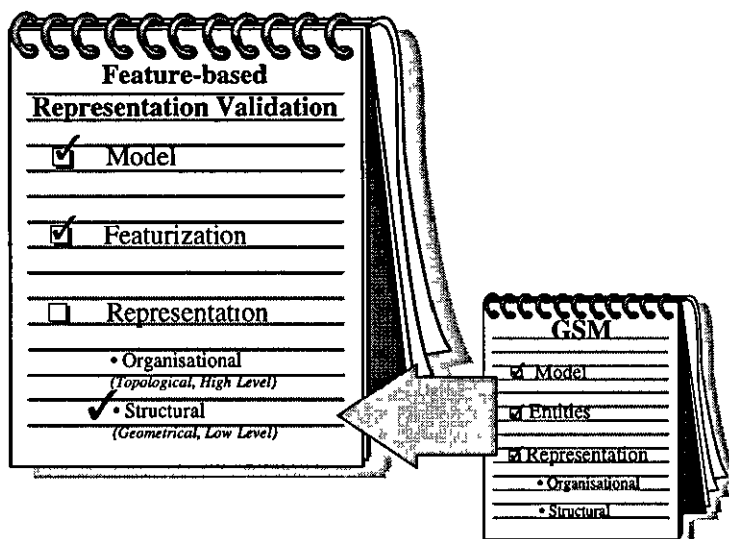


Figure 3-1: Organisational and Structural Validation Levels.

- The other organisational or semantical validity conditions are termed hereafter *Conceptual Validity Conditions* because they are concerned with

the feature's *concept* (their role and semantics as a 3D modelling technique, their expected behaviour and their high-level organisational meanings).

3.2.2 CONCEPTUAL FEATURE VALIDATION

Features play an important role in capturing the designer's intent in computer-aided design (CAD), raising the abstraction level of geometric design and facilitating integration with applications such as computer-aided manufacturing (CAM) and computer-aided process planning (CAPP). However, such integration will only be profitable if the feature model is valid in terms of the feature's concept or functional meaning. For instance, if a *pocket* (or a *blind hole*) in the model is allowed to pass through the part, this misrepresentation could cause machine damage, mistakes or, at least, non-optimised decisions by a CAPP system.

Feature-based validation allows CAD systems (usually more preoccupied with representing and producing feature-like shapes within a geometrically constrained environment) to interface more easily for example with CAPP systems (usually more preoccupied with planning problems than with the correctness of the representation).

As DbF systems usually subsume an implementation on top of a GSM scheme (such as CSG, B-rep or hybrid), then they also subsume the availability of low level modelling operators (such as Euler or Boolean Operators) as well as GSM validations (such as Euler-Poincaré formulae verification or Boolean regularisation, Zeid91). These low-level operators are not included in this study. Other approaches (Stroud93, Subrahmanyam95) go into this level.

Conceptual feature-based model *representation validation* thus implies that the verification of the intended functionality of a given feature must conform with the geometric semantic meaning assigned to that specific feature type.

Conceptual Feature Validity Conditions may be translated as reasonings and 'enquiries to the underlying GSM as well as to information stored into the Feature Modeller, alone or altogether' (Rossignac90).

Like structural validity conditions, Conceptual Feature Validity Conditions may have to be evaluated frequently because many common manipulations can lead to a valid solid representation but not to a valid feature-based representation. A DbF system that allows the use of high level entities such as features to represent abstract concepts of designer's intents should guarantee that this representation is valid and should reason using a vocabulary of the same high and abstract level, the conceptual or semantical level. The feature-based reasoning should use mainly feature types, descriptions and parameters (rather than their geometrical B-rep or CSG evaluations) as a 'vocabulary' for validation analysis and manipulation operations.

Structural, low-level and syntactical representation validation has received considerable attention in the literature (see validation facet types from sections 3.1.1, 3.1.2, 3.1.4, 3.1.5 and 3.1.6).

Work conducted by Pratt, Duan, Bidarra, Martino, Su, Shah and Zhang and respective colleagues is considered to have recognised feature-based *conceptual* representation validation to some extent.

3.2.3 VALIDATION AT FEATURE CLASS LEVEL

Denzel93 argued that there is still scope for reducing programming effort in defining new features. An approach that deals with feature validation regardless of the feature type would help ease that effort. The term conceptual feature-based validation suggests an approach in this direction: the validation of the feature's concept as a whole and not of its particulars.

Therefore, generic validation reasonings that can be applied to all feature types are of major concern in this research. In approaching the problem in this way, the task of defining totally new features (which greatly enhances the system flexibility) is facilitated because few extra behaviour and validation activities need to be defined and programmed.

In an object-oriented approach it could be called a feature-class reasoning because it is defined for the feature class and all the objects (feature types)

derived from it (such as a *slot* and a *hole*) would inherit the behaviour and validation reasonings of the class.

3.3 FEATURE VALIDATION SYSTEM

Although there are no universally applicable methods for checking the validity of features, three elements were identified as necessary to compose a conceptual feature-based representation validation system (see Figure 3-2).

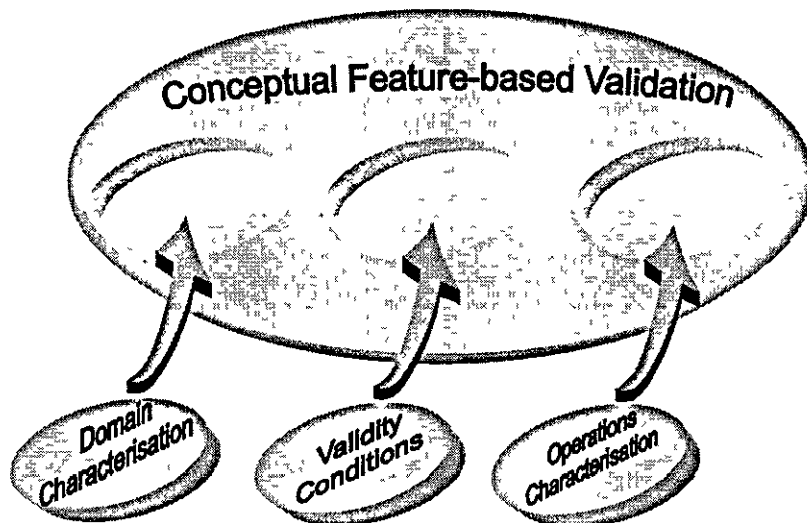


Figure 3-2: Conceptual Feature-based Validation Framework.

3.3.1 DOMAIN CHARACTERISATION

To perform conceptual feature-based representation validation it is necessary to establish properties with which a representation must comply. This is done by a proper *domain characterisation* that reflects and makes explicit some common-sense feature behaviours as a 3D modelling technique. The *domain characterisation* should produce a set of properties that are intrinsic to feature technology and to the idea of using features as a modelling resource. In addition, these properties should be made measurable otherwise they would not contribute to the automatic validity analysis of the feature-based model.

3.3.2 VALIDITY CONDITIONS

Another element of the validation framework is the set of conditions that are produced to assess the conformity of the representation with the properties that characterise the domain. These are called *validity conditions*. Validity conditions are important because the richness of the model assessment dictates the success of the representation validation. For instance, it has been said that the capability of checking various types of technical validity criteria is critical for using features in manufacturing planning (Shah95). Therefore, the conditions should consider a great variety of situations for each property being analysed.

3.3.3 OPERATIONS CHARACTERISATION

Considering the *domain characterisation* and verifying the model with *validity conditions* allowing the identification of invalid and valid representations is an advantageous aspect of a feature-based system. However, it is possible that a better understanding of the invalidity phenomena could come from a better understanding of their origins: *operations* and the consequent *feature interaction* phenomena. Therefore, a characterisation of the operations available in the system is the third element of a feature validation system framework.

In particular, for pragmatic reasons, an enormous advantage could be achieved if operations capable of correcting invalid representations can be identified.

For the example shown in section 1.5.4 these operations for invalid B-rep solid representations (*revalidation operations*) include:

- elimination of dangling faces and edges;
- performing “regularisation”;
- unification of vertices referencing the same point and then trying to validate the result;

- re-orienting a face's edge loop;
- eliminating NULL pointers from the data structure.

There are no similar well-defined *revalidation operations* in the feature-based context, and therefore further research on this topic is required. These *revalidation operations* could be automatically invoked or offered to the designer by the system according to the invalidity case.

3.3.4 THE FRAMEWORK

To support conceptual feature-based representation validation a system must be built upon a thorough definition of the constituent elements (Figure 3-2). The very definition of features and their characterisation should be made in such a way as to be suitable for verification and to be in accordance with expected common sense behaviour of features. A volumetric analysis of features seems to be an adequate (feasible) candidate for this purpose.

Further, if the characterisation formalism is made clear, verifiable and representative enough the system could perform automatically the identification of complex relationships between features. This automatic recognition will promote the designer's freedom from this tedious task and the enrichment of the representation. However, the human understanding of the model and her/his intervention will be necessary to accept or reject a recognised relationship as an important and desired one. Moreover, once these relationships are as meaningful as features are, this process could possibly drive a more efficient modelling environment.

3.4 DETAILED RESEARCH SCOPE

In this research volumetric features have been implemented in a Hybrid CSG/B-rep solid representation scheme where features are closed sets of boundaries and Boolean operations are available. The aim is to perform

feature-based model *representation validation*, and model and entity (featurization) validation are outside the scope of the work.

This interest in the conceptual representation demands that a Design-by-Features approach is adopted where features are instantiated from a library rather than recognised from a geometric model. The validation analysis discussed here considers that all features are in the same (form) feature representation space and therefore no conversion or mapping procedure is performed or considered.

Procedural modelling is used in the sense that the CSG-tree is traversed and re-evaluated when required, thus removing the need for facilities for localised updating of the GSM data. Neither parametric nor geometric constraint-driven environments were used.

Orthogonal positioning is used, i e. features are placed parallel or perpendicular to one of the Euclidean axes. Prismatic features related to milling and drilling are the main concern. Tolerance conditions have not been considered. Stock material is limited to a rectangular block.

Model building is achieved with *add* and *delete* operations and no access to low level geometric or topological entities is required for editing operations. Hierarchical representation/modelling is not required and *parent-child* relationships are not forced.

3.5 WHAT TO VALIDATE ? FEATURE-BASED DESIGNER'S INTENTS

It has been alleged that designer's intention is not always described explicitly in the result of a geometric design using current (conventional and GSM) CAD systems (Yoshikawa87) and that "features are expected to reflect the designer's thoughts" (Lenau93). FBM represents a step forward to overcome this limitation. However, capturing design intent is difficult as it can be extremely variable and unstandardised. Also, very few feature types and implementations are available at the high abstraction levels (closer to the conceptual design stages).

Designer's intent has been considered part of a validation method and even considered essential in decision-making where different kinds of interference happens among features (Zhang93). This is so because interference can be valid in one application but invalid in others. In addition, capturing and storing designer's intents in the part model make tasks such as feature modification much easier (Sheu93).

Features themselves have been claimed to be a convenient vocabulary for formulating validity checks that assess the compliance of the model with the designer's intent (Rossignac90) because features widely capture designer's intent provided that their semantics are oriented to the application (Gomes91). On the other hand, there is some confusion between a feature's own functionality and the functionality of various features acting together: Rimscha (Rimscha90), has classified features into two categories: *free features* which are geometric form features that compose the part and *embedded features* which have no existence on their own but which establish all sorts of relationships between *free* features or assembled parts and *embedded* features were said to occur at various levels of the part representation (Nnaji93, Rosen93).

Features are intimately related to designer's intent. However, although it has been claimed that FBM systems capture and represent designer's intents to some extent (Dixon87, Rossignac90, Silva90, Rimscha90, Nielsen91, Rossignac91, Zhang93, ElMaraghy93a ElMaraghy93b, Su94, Shah94b, Chen95, Taylor96), few attempts (Dixon90, Henderson93, Nnaji93) were found to define, clarify and identify designer's intents within the feature-based modelling context.

Therefore, a *domain characterisation* that leans towards conceptual feature-based representation validation should be achieved by defining *Feature-based Designer's Intents (FbDI's)* that express feature behaviour.

It seems difficult to conceive at first that another vague and abstract term, namely Designer's Intents, could be the measurable means to perform

conceptual feature validation. Nevertheless, the absence of a formal mathematical definition for features and a consensus that they are the carriers of designer's intents make this option look more inviting.

3.5.1 DEFINITION

Designer's intent's are of "high importance to be preserved but their understanding has a complicated nature" (Yoshikawa87). Some of the clarifications for designer's intent in the context of feature-based modelling include:

- 'Design intents state the purpose of an aspect or underlying rationale behind an object' (Henderson93) that help justify a design decision. For instance, a *hole* could act as an oil outlet or a fixing point, which later will help decide on machining tolerances.
- 'Designer's intents are the feature's reason for being in the design and hence, the reasons certain dimensions and spatial relationships are what they are' (Dixon90).
- 'Designer's intents are a set of functions which the product will provide or require' (Nnaji93).

It has been acknowledged that "the information that constitutes intent, and how to capture and use intent are all research issues to be explored" (Dixon90). Thus, it is herein defined that

Feature-based Designer's Intents (FbDI's) represent a variety of concerns that help decide on a specific feature attribute or configuration. They are factual peculiarities of the geometric design that are intrinsic to features themselves or to the use of features in the design and have engineering-related purposes. FbDI's are properties that are expected to arise in the model because of the use of a feature in a specific location or because of the interactions that a feature provokes with the existing surrounding features in the model.

FbDI's represent information that should be verified and maintained throughout the detailed design process and could be used as restraints to drive the decision-making process of a downstream application. Because they are considered intrinsic to features, they are sometimes left out of the formal and explicit description of a design.

Their presence could be easily overlooked but their absence is immediately recognised by the designer. For instance, a series of *hole* features on a part in an intentional pattern suitable for a optimisation could be overlooked but if they are not in such a pattern, no optimisation of that kind would ever be tried. Either presence or absence affects engineering-based reasonings and computer-aided systems such as CAPP, CAM and CAE.

3.5.2 WHICH FBDI's ? VOLUMETRIC FBDI's

Despite any implementation issues, it is common sense that form features have a strong volumetric meaning (Pratt85, Pratt88, Shah88e, Dixon90, Li90, Rossignac90, Perng90, Rosen93, Tseng94) and usually their expected behaviour as shape builders is guided by their volumes. Therefore, Volumetric Feature-based Designer's Intents (VDI's) are concerned with the feature's expected volumetric behaviour, or *volumetric intention*, which is comprised of a feature's *nature* and a feature's *volume*:

- A feature's *volume* (FV) specifies that there is an intention to imprint a specific volume or shape onto a part. It has also been called a *Feature Producing Volume* (FPV, Pratt85), *Volumetric Cell* (Gomes91), *Self-Contained Volume* (Gindy89, Su94), *Basic Volume* (Mayer94) and *Feature Associated Volume* (FAV, Bidarra94).
- A feature's *nature* (FN, Lenau93, Kraker97) specifies that there is an intention of *adding* material (when it is said to have a *positive* volume) or *removing* material (when it is said to have a *negative* volume) from the component.

FV and FN (see Figure 3-3) together define how to imprint a specific shape onto the component. FV's do not need to have all the explicit evaluated surfaces that represent the FV in the evaluated feature model - the component (as suggested by Rossignac90 for Intentional Features, see section 2.7).

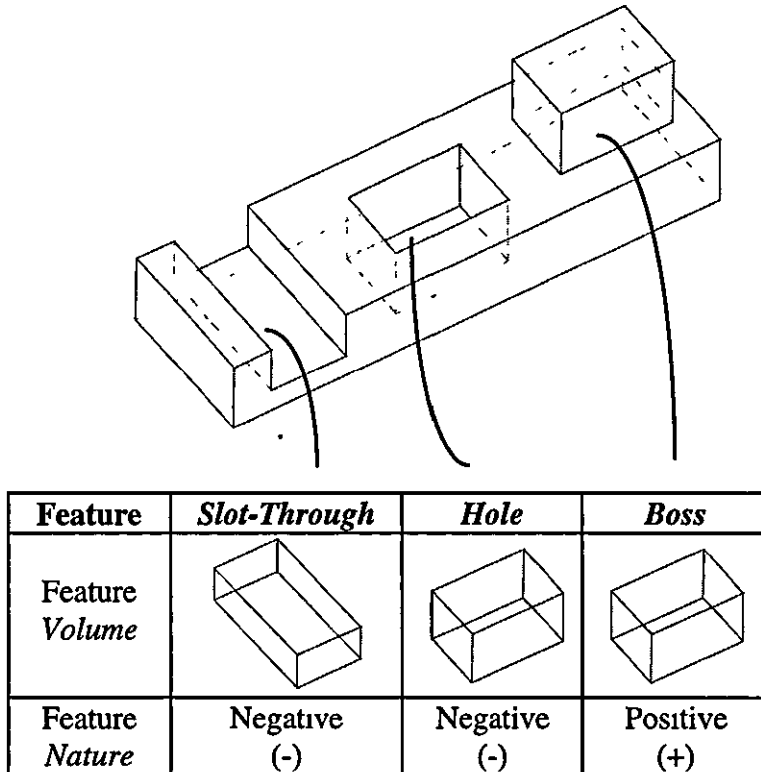


Figure 3-3: Examples of a Feature's Nature and Volume.

Figure 3-3 presents examples of features and their respective *volumetric intention*, FV's and FN's. It should be noticed that, apart from the orientations, the FV's are very similar and only when positioned and with FN's established is a specific feature type achieved on the component's surface. This shows the importance of *volumetric intentions* to FBM.

The volumes associated with a feature are so important that not only the actual feature volume has been used but the whole removing and clearance volume swept by a tool during the process of manufacturing a feature has been considered and this has been called 'manufacturing motion feature' (MMF, Medland93). Also, feature volumetric intentions have proved to be very important resource for planning and machining optimisations (Li90, Tseng94)

Thus, the Conceptual Feature Validation concerned here is biased by a volumetric interpretation of features although it is applicable to surface feature implementations and in fact, also includes some boundary reasoning.

Volumetric FbDI's are considered particularly important when an interaction occurs between feature *volumes*. To deal with VDI's, the semantics of non-conflicting and conflicting volumetric interaction between features have to be defined. This analysis led to the identification of four VDI's (see Figure 3-4): Changeability, Fittability, Redundancy and Labelling.

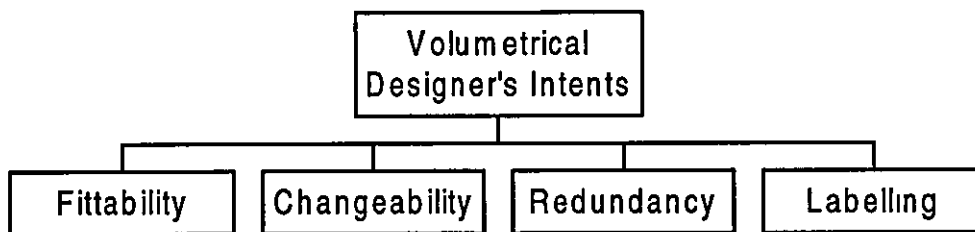


Figure 3-4: Volumetric Feature-based Designer's Intents.

3.5.2.1 CHANGEABILITY VDI

A Feature's *nature* implies that a change in the feature-based representation must result in a change in the volume and surface of the component being modelled. This feature requirement and ability to change the existing model is called the *changeability* VDI. However, it does not require that all the boundaries of the FV should be shaped into the part. The changeability requirement invalidates *obsolete features* (Shah90) that occur when a feature is completely inserted into another and has the same *nature*.

Obsolete features have no functional significance to the model and this should be acknowledged by downstream applications or they should be eliminated in the first place.

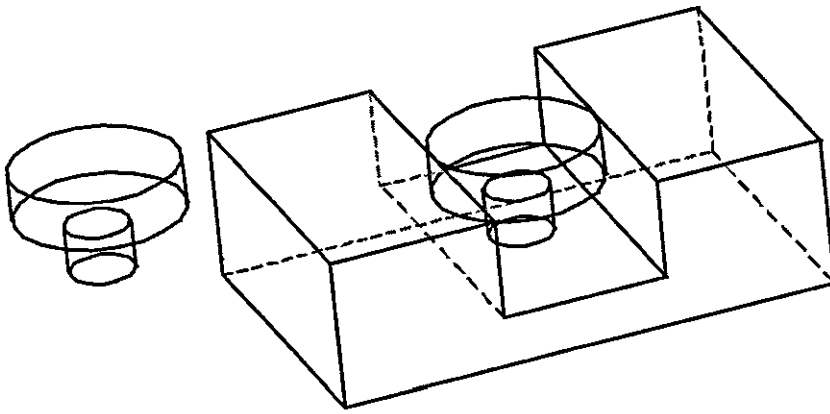


Figure 3-5: Invalid Representation Containing Two Obsolete Features.

Figure 3-5 shows a feature-based representation with two counter-bored *hole* features, one placed disconnected to the stock material and the other placed inside the removing volume of a *through slot*. Both *holes* are considered obsolete because their volumetric intentions do not affect (change) the actual representation of the part. Therefore this is an invalid representation.

3.5.2.2 FITTABILITY VDI

A feature must have adequate parameters to properly express the extent of its underlying intentions and functionality. Thus the feature must fit within the limits where it is intended to be placed (in the same way as an *edge* is limited by its two exact ends, called *vertices*, in B-rep). This aspect is called the *fittability* VDI.

The fittability requirement invalidates the problems of *feature's parameter made obsolete* (partially considered by Shah90) where a feature's parameters do not describe exactly the extent of what it imprints on the part and two cases can be identified (see Figure 3-6):

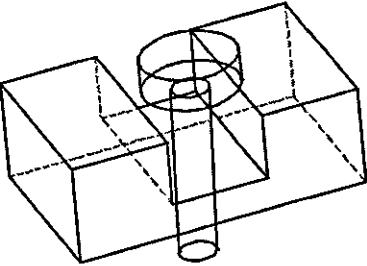
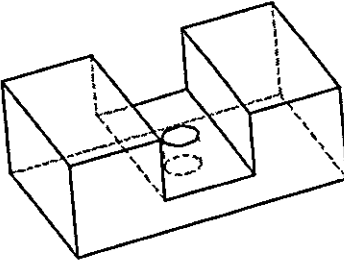
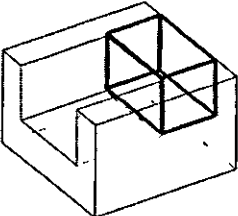
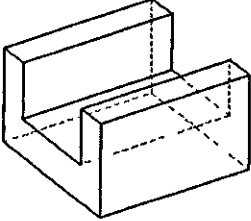
	Invalid Representation	Validated Representation
Parameter is too Large		
Parameter is too Small		

Figure 3-6: Examples of the *Fittability* VDI Problem.

- *Parameter is too large* such as when a feature exceeds the limits of the stock material.
- *Parameter is too small* such as when two features touch each other with a perfect face *match* such that they can be *united* or replaced by another feature that encompasses both behaviours.

However, adequate parameters could also mean that part of the feature does not affect the component and hence, the *hole* in Figure 3-7.(a) is a valid representation as a single *through-hole*.

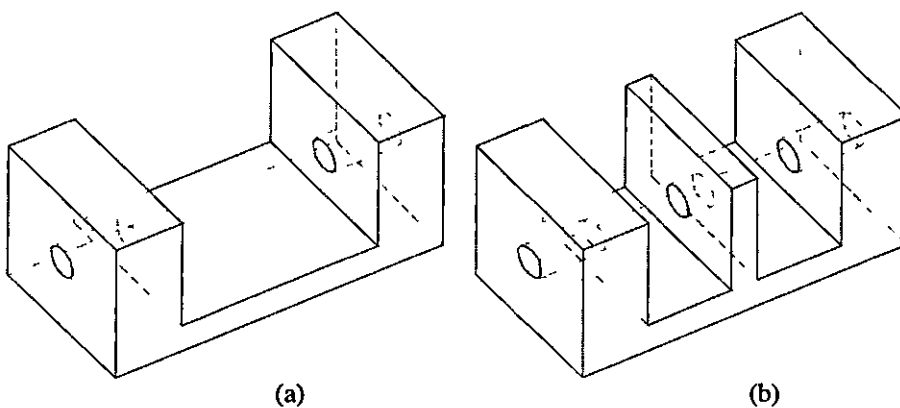


Figure 3-7: Volumetric FbDI's Analysis and Adequate Parameter.

3.5.2.3 REDUNDANCY VDI

Interesting and difficult situations arise when redundant *volumetric intentions* are found. Consider a part (Figure 3-7(a)) comprised of a *slot* and a *hole* (modelled as a single long cylinder). There is a *redundancy* of VDI's where the *hole crosses* the *slot*. This is a feature interaction problem that has been receiving much attention in the literature as being of special difficulty to handle (see chapter 5). Nevertheless, concerns are drawn here regarding the *redundancy* VDI management and not the feature recognition, extraction or even internal representational problem.

Thus, whether to accept the feature redundant *removal* intent in the representation or to *split* the *long hole* into two *short hole* features has consequent implications:

- If the redundant portion is allowed, care should be taken during further analysis to avoid redundant manufacturing operations (for example).
- On the other hand, if two separate *short hole* features are created, the redundant geometric part must be *deleted* from the model and other types of intents (such as the *equal radius* and *concentricity* intents between the two *short holes*) must be added to the representation.

However, eliminating the redundant part also eliminates the removal intent at that location. For instance, consider now adding a *boss* into the *slot* (Figure 3-7(b)). Should this *boss* have the former *hole* intent as well? Or, consider removing the *slot* if the second approach is taken: what happens to the formerly deleted and redundant part of the *hole* feature?

Capturing and maintaining VDI's is a subjective problem. For instance, simple operations such as adding a boss inside a *slot* (Figure 3-7(b)), could be interpreted as an acceptable situation (when it leads to the formation of a protrusion for example) or could be interpreted as a VDI conflict since it is precluded by a material removal intent. Inverse operations (superimpose a material removal over an additional intent) can cause hollows/cavities (usually

undesirable) or the loss of (part of) the initial addition intent (deleting the previous feature at that same location).

Similar FbDI management problems appear when simple *delete* operations are required. Imagine the volumes used to produce a component formed by a *step*, a *slot* and a *hole* feature (Figure 3-8(a)). Now imagine re-adding the volume that produced the *slot* as a way to delete the *slot*. Not only is the *slot* deleted but also the *step* is turned into two *notches* and the shape of the *hole* is affected. A clearly unwanted VDI scenario has emerged instead of the simple deletion of the *slot*. This is a matter of managing the redundancy VDI. If the system was able to identify the portions of the feature that represented a redundant material removal procedure then it would have known that the deletion of that part should not be made by just re-adding material, although for cases of non-redundant intents re-adding the initial volume suffices.

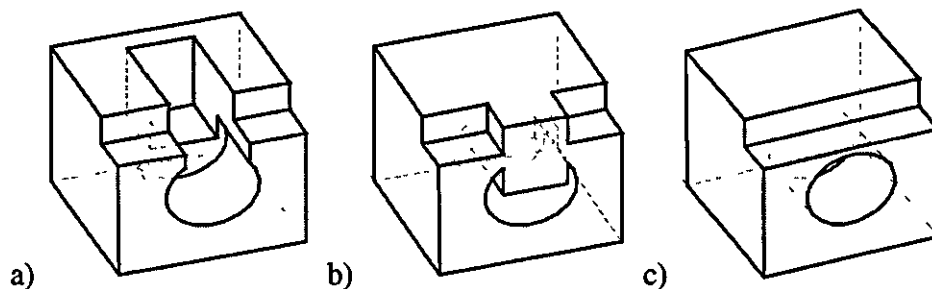


Figure 3-8: Deleting Volumetric FbDI's.

A proper *delete* operation could be achieved if the history of the design was stored allowing the later reconstruction of the model. However, high computing time will be required to reconstruct the model after every manipulation, and thus there are some approaches that perform “local updates” on the model to save computing time (Rossignac90, Gomes91, Sheu93, Vandenbrande93).

3.5.2.4 LABELLING VDI

The *labelling* VDI identifies the relationships between all feature faces and their attributes. Every feature has a set of relationships that is kept as the feature's *label* and identifies features as being of a specific shape and having a unique behaviour. *Labelling* is basically defined by a template of *virtual* and

real faces that wrap the feature *volume* of a feature type (see Figure 3-9). *Virtual* faces basically identify tooling external access directions and *real* faces identify surfaces to be imprinted on the part.

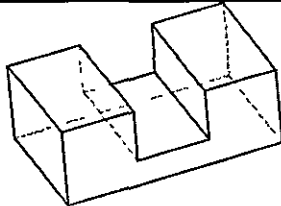
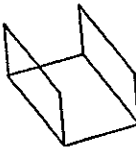
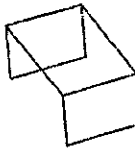
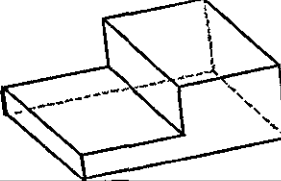
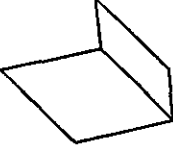
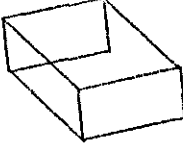
Feature Label	Feature Shape	<i>Real</i> Faces	<i>Virtual</i> Faces
<i>Through Slot</i>			
<i>Step</i>			

Figure 3-9: Feature's Template of *Virtual* and *Real* Faces.

It should be noted in Figure 3-9 that the *step* and *through slot* features have a very similar volume (FV) and the same negative nature (FN) and therefore a very similar *volumetric intention*. However, the set of *real* and *virtual* faces helps identify the proper *label* for each case, which leads to very functionally different features.

A feature's *nature* and *volume* are closely and complementarily related to the feature's *label* (its positioning and template of *real* and *virtual* faces). A *label* is considered to be the link between the geometric modelling realisation of the feature-based model and other non-geometrical information associated with a specific feature type.

For instance, the same *nature* and same location but slightly different template description would result in a different feature *label* (such as for *through slot* and *step* features, see Figure 3-9). On the other hand, a feature's *label* is dependent on the feature's positioning and thus changing a feature's position could possibly change the feature *label*.

If a face of a given feature *abuts* and is completely *inserted* into another feature's *real* face, the former must be a *virtual* face (Silva90). Using reasonings such as this the labelling aspect can be maintained (refer to section

8.9.3). If the template and the realisation do not match, the feature *label* is invalid and a “revalidation” operation of searching for the appropriate *label* should be invoked.

3.5.3 WHAT ARE FbDI'S ? THE ESSENCE

To continue arguing that FbDI's can be explicitly captured and manipulated, a sensible internal representation needs to be found. This raises the question of what are FbDI's in practical terms?

FbDI's have been considered to be constraints by many authors (Dixon87, Nielsen91, Dohmen96) because the resulting geometric relationships are set by the designer in a conscious manner to express some of the design objectives. However, the way a network/graph of constraints is set up is particular to a specific design or application, and therefore it will be impossible for the designer to generalise the constraints to cover all possible situations of similar designs (Zhang93).

Furthermore, it has been alleged that a constraint-based system is neither sufficient nor easy to work with (Rossignac90) for a design application. In addition, constraints were regarded as unable to accommodate all sorts of associations (Hailong95) or relations (Henderson93) in a design. Therefore, it is difficult to accept that a very specific set of relationships can be regarded as all possible FbDI's that exist in a design. Besides, these constraints have been established using geometry information rather than feature-based information.

Because of these considerations, it is believed that constraints are part of the FbDI concept and thus the FbDI concept must accommodate constraints in some way. Furthermore, considering that FbDI's have also been regarded as connectivities between features and as flags for intended interference (Zhang93), FbDI's are defined as general relationships between features (preferably) and/or feature elements (thus, accommodating geometric constraints).

3.5.4 THE ACTUAL VDI'S

The following are the FbDI relationships originated from the volumetric FbDI's reasoning:

- A feature identified as *merged_from* another two features originates from a *fittability* VDI analysis.
- A feature identified as *split_into* two or three other features also originates from a *fittability* VDI analysis.
- A feature identified as *obsoleted_by* another feature originates from a *changeability* VDI analysis.
- A feature identified as *deleted_by* another feature originates from a *redundancy* VDI analysis.
- The *label* of a particular feature (one of its properties) originates from a *labelling* VDI analysis.

3.6 EXAMPLE OF CONCEPTUAL MFI REASONING

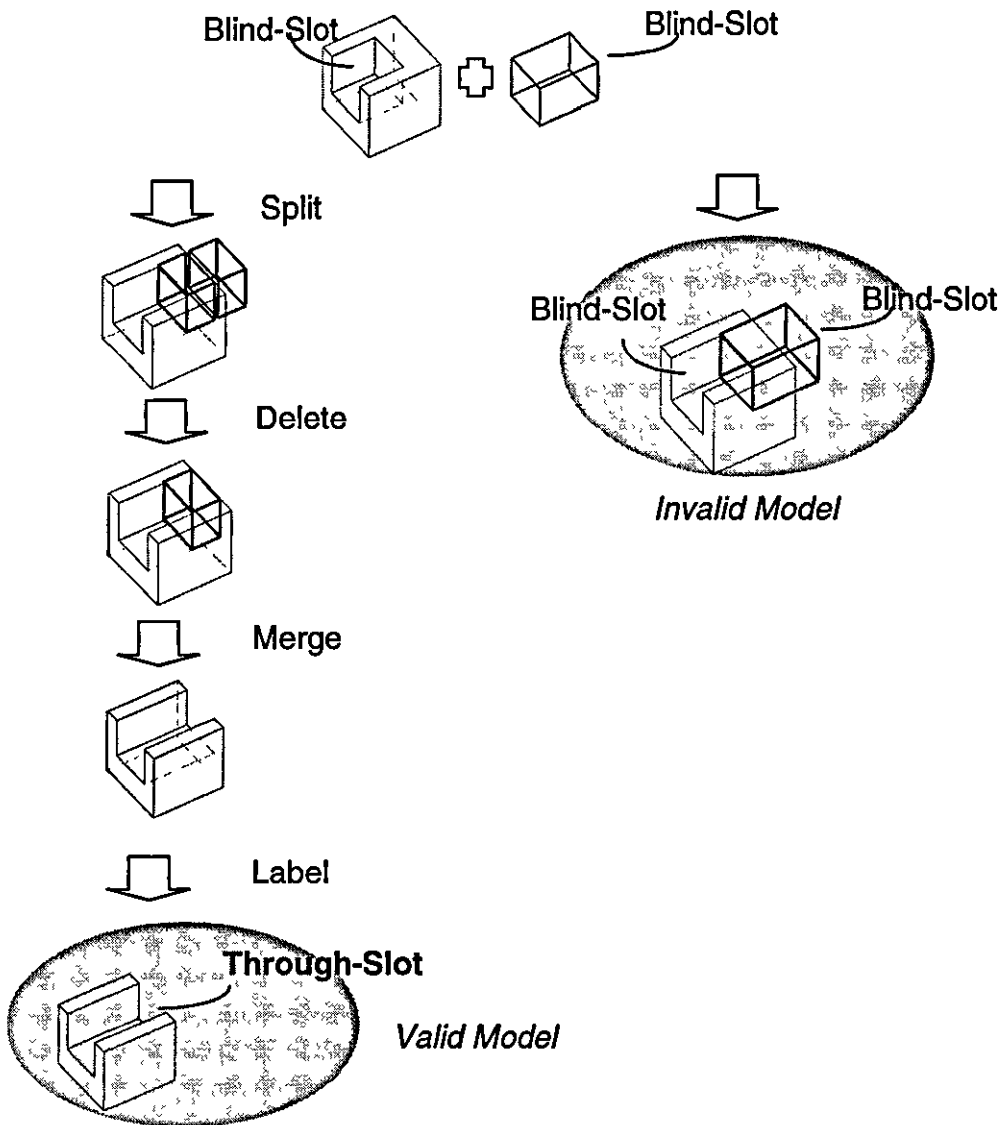


Figure 3-10: Example of Conceptual MFI Validation.

Figure 3-10 presents an example of conceptual representation validation reasoning and some of the operations involved in this process. The right-hand side of the figure shows how the model would appear if no validation reasoning is applied, while the left-hand side shows, operation by operation (see chapter 6), how a conceptual representation validation reasoning can be applied in order to obtain a valid model at the end of the process.

3.7 COMPOSING A FEATURE VALIDATION SYSTEM

Validity conditions should be devised to verify the semantics of the *domain characterisation* (Volumetric FbDI's). This can be achieved through a suitable vocabulary to express the conditions regardless of whether they are to be implemented in a Knowledge-based System (KBS) or in a C-like language. It can be inferred that three elements influence the way VDI's have been presented and explained, and therefore should be used to compose such a vocabulary. These vocabulary elements are:

- The way that features interact (e.g. *inserted face* and *cross feature*).
- Feature attributes (e.g. feature's *nature*, face's *virtual* or *real* attribute).
- The way that features are operated on (e.g. *delete* and *add* a feature, *split* feature and *search for new label*).

Furthermore, some ways in which VDI reasonings relate to other types of intents (e.g. *co-radius* and *concentricity*) have already been mentioned (section 3.5.2.3) and can be created as a consequence of a VDI reasoning.

This makes it clear that there are non-VDI intents, that these other types of intents also need to be properly defined and specified and that VDI reasoning is not an isolated reasoning. This suggests that conceptual feature validation can be extended to a much broader spectrum of designer's intents becoming an intent-driven approach for reasoning about feature-based models.

3.8 SUMMARY

Modelling and editing feature-based models produces models that potentially violate the feature's expected behaviours. A step of verification and validation is therefore essential, although what to validate is still open for interpretation. This validation process is difficult because the extra information, or *designer's intents*, that features carry is not easy to formalise and quantify.

Conceptual feature-based representation validation requires the definition of feature *volumetric intentions*. These terms, *intents* and *volumetric intentions*, have been identified and specified. These reasonings arise from the elements of a vocabulary. This vocabulary includes feature interaction, intents interaction, feature attributes and operations. The elements that comprise a system capable of analysing a representation against such volumetric FbDI's have been identified.

When features are manipulated or introduced into a part's representation they become part of a complex arrangement of entities, which are initially isolated, but which subsequently interact with each other. Therefore, two intrinsic dimensions of information identify a feature before and after being incorporated into the model: *intents* and *interaction*. Feature interaction seems to be useful in testing, validating and correcting feature-based models from the *designer's intents* perspective.

In the following chapters these vocabulary elements (feature-based designer's intents, feature interactions and feature operations) are presented and properly classified, typified and specified to comprise not just a validation system but also an intent-driven reasoning approach.

4. AN INTENT-DRIVEN APPROACH

Features have extra non-geometrical semantics and have been considered to carry designer's intents which are used by many applications but never related back to these designer's intents. This chapter presents a classification of designer's intents and, as they are considered to be properties that are intrinsic to feature-based modelling, they will be used as the means by which validation conditions are set. Therefore, adopting the approach of defining a taxonomy of designer's intents helps define the role of features in geometric design and, indeed, allows future feature-based modelling systems to better represent, store and reuse such information. Moreover, it allows a more formal approach to manipulating, verifying and maintaining designer's intents throughout the design process, which is invaluable support for genuinely intelligent CAD systems.

4.1 CONCEPTUAL VALIDATION AND BEYOND

Although features are a proclaimed and accepted means of capturing and representing feature-based designer's intents (FbDI's), existing systems do not deal with FbDI's as their major concern. The main reasons for this are threefold:

- first, there still is a lack of a formal well-accepted definition for features and their role as a geometric modelling technique.
- second, there is the same lack of understanding of what FbDI's are, especially in the context of FBM.
- third, identified intents are usually blended, immersed or diluted within the application under consideration.

Capturing FbDI's at early stages of the design in a more user-friendly interface that includes a vocabulary meaningful to the designer is a property of a design-by-features (DbF) system that allows more intelligent decisions and reasonings to be made and has been considered as a necessity for Intelligent CAD (Cunningham88, Dixon90).

A validation system need not be used solely for conceptual validation and the associated *volumetric intention* reasoning. It could also be extended to validate various other types of intents and therefore become an intent-driven reasoning system. To identify and understand these other types of FbDI's an entities *elicitation process* (explained for feature elicitation in section 1.5.2 and summarised in Figure 1-8) is suggested.

Depicting all sets of FbDI's present in the designer's mind is beyond the scope of this research and is a very cumbersome approach even in a limited domain. The objectives of this chapter are to (a) explicitly categorise FbDI's in such a way that this extra information could be effectively and consciously instantiated into a model, and (b) provide a set of characteristics that a DbF system could be based upon. In this way, the capturing, verifying and maintaining of FbDI's could be performed by, and even automatically discovered by, a DbF system.

Moreover, this approach provides another dimension to be considered when designing and implementing FBM systems as these would be constructed upon the concept of intents to be achieved and validated, and not just the set of features to be manipulated.

4.1.1 FEATURE-BASED DESIGNER'S INTENTS ELICITATION

As discussed earlier (section 1.5.2), the *elicitation* process requires the characterisation of a *domain*. A process of identification based on *elicitation criteria* follows and this can be helped by an appropriate *classification*. A *taxonomy* is then produced considering an application and another set of criteria validates a (possibly smaller) set of elements for a chosen *application domain*.

The *domain* adopted for this research is the integration of feature-based CAD and CAPP and information related to it has been mainly gathered from related publications, and therefore FbDI's presented are the ones perceived from these systems.

4.2 DESIGNER'S INTENTS ELICITATION CRITERIA

A reasonable set of "manageable" FbDI's should be clearly identified and classified to match feature semantics and this is achieved via a suitable set of *elicitation criteria*.

Keeping a pragmatic awareness of the implications for DbF implementations, the following set of *elicitation criteria* were established to select objective, concrete and verifiable FbDI's:

- they must have importance to the decision-making process of detailing a geometric design and hence, are *not* for *documentation* or illustrative purposes solely. "Designer's intents" that do not usually constitute a representation of the design knowledge to be used for subsequent verifications or to trigger reasonings should be avoided.
- they must have *geometric semantics* expressed in a way that is suitable for association with features and for building a reasoning process.
- they can be of *hierarchical* nature, where high-level more abstract FbDI's can be defined, but there are basic atomic FbDI's which are preferable.

- major attention should be paid to FbDI's that are *computable and inferred* during the design process rather than to those that can only be explicitly stated by the designer (Silva90, Dixon90, Suzuki90, Zhang93, Vancza93, Salomons93, Mill93). This does not mean that this process is easy or already available but does mean that it is conceivable.
- FbDI's that build a hierarchy with tight dependency should be avoided or kept as a distinct class to maintain simplicity of the reasoning (see Parametric FbDI's, section 4.4.2).

4.3 DESIGNER'S INTENTS CLASSIFICATION

A feature model is considered invalid if it does not fulfil its functions (Martino94a). Three types of FbDI's have been identified (see Figure 4-1): morphological functional, theoretical functional and relational functional.

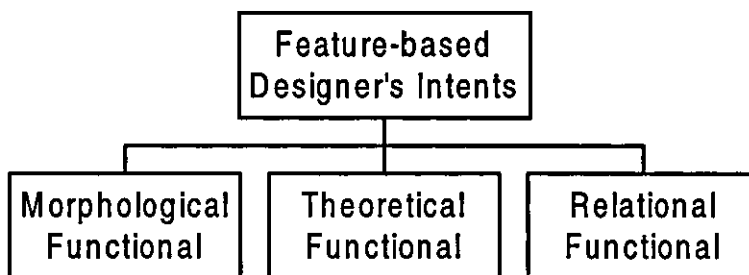


Figure 4-1: A Classification of Feature-based Designer's Intents (FbDI's).

4.3.1 MORPHOLOGICAL FUNCTIONAL FBDI (MFI)

Features represent a good means to embed functional significance into the geometric detailed design phase and this fact can be inferred by some definitions applied to features. Features have been defined as:

- the addition of functionality to geometric forms (Dixon90, Sodhi91, Nnaji93);
- high-level morphological information with well-defined functional meaning (Gomes91);

- high-level functionally significant entities (Laakko93, Bronsvoort93).

In addition to topological and geometrical analysis that is usually applied to identify features (as in Feature Recognition approaches), extra functional factors have been added to better specify the elements of a feature family.

For instance, a *cylindrical boss* family of features could be specialised into a *disk* if limited to a certain height-to-diameter ratio range; otherwise, it becomes a *rod* (Nielsen91). In addition, a *hole* feature can be assigned as a morphological functional FbDI but it will only be categorised as *drilled hole*, *bored hole* or *punched hole* when the application domain is considered (in this case, manufacturing capabilities).

This functional specialisation generates a drastically different manufacturing approach. In the case of the example above, it could be machining the *disk* or welding the *rod*.

It has even been considered that, if an application considers only functional morphological information (shape) then the term “form feature” can be used (Dohmen94). These considerations clearly expose features as having *morphological functional FbDI's (MFI's)*.

4.3.2 THEORETICAL FUNCTIONAL FBDI (TDI)

Features are also linked to the *function* concept itself which has been defined as “the behaviour of an object, an operation of energy, material, information or signal that tells what the design does” (Tomiyama93) and “include not only in-use purpose, but also manufacturing and life-cycle considerations’ (Dixon90).

Although some researchers have addressed the relationship between form and *function*, it is not formally understood yet because of many difficulties (Shah90, Salomons93).

- firstly, the abstract nature and understanding of the *function* concept.

- secondly, functionality can be a composite result of many interacting sub-*functions*.
- thirdly, a given *function* could be performed by several forms and one form could be used to perform different *functions*.

Functions make specific shape aspects appear on the part's surface, control the part's overall outlook and are driven by a close relationship between a feature's theoretical functional behaviour and its form. This is possible by manipulating and controlling the hierarchy or dependency of parameters that establish dimensions, profiles (e.g. quadric, circular, spherical), parameterised local operations (blending, chamfering, trimming) and so on.

This *function* concept has been implemented as physics-based or engineering-based laws, rules or formulae depending on the underlying theory, such as heat propagation, torque or force transference or stress analysis (Taylor96). Thus, they are called *theoretical functional FbDI's* (TDI's).

4.3.3 RELATIONAL FUNCTIONAL FbDI (RDI)

While TDI's are usually expressed by formulae, engineering constraints are expressed in the form of relationships between entities. Thus, they are called *relational functional FbDI's* (RDI's).

RDI's comprise different disciplines and are dependent on the application of the feature-based model. RDI's are mostly geometrical facts that have a functional significance for an application.

For instance, a "*nested at the bottom*" relational FbDI (see section 4.4.3.1) is a geometry-based and provable fact that could be used by a CAPP system to establish machining precedence among features.

4.4 DESIGNER'S INTENTS TAXONOMY

Each FbDI type has a set of objective and tangible properties at a “pragmatic” level, which helps to implement FbDI's within the geometric realm. FbDI types specify general engineering concepts or behaviours while the actual FbDI's are computable relationships between features themselves or elements of the feature-based model such as feature faces (and their attributes) and feature parameters, and are to be implemented in a system.

The FbDI's presented below comply with the *elicitation criteria* presented before and thus, are verifiable, measurable and manageable. This enumeration of FbDI's can be said to build up a *taxonomy* of FbDI's.

4.4.1 MORPHOLOGICAL FBDI

Morphological functional FbDI's (MFI's) can be achieved through Volumetric FbDI's (VDI's) introduced and detailed in section 3.5.2.

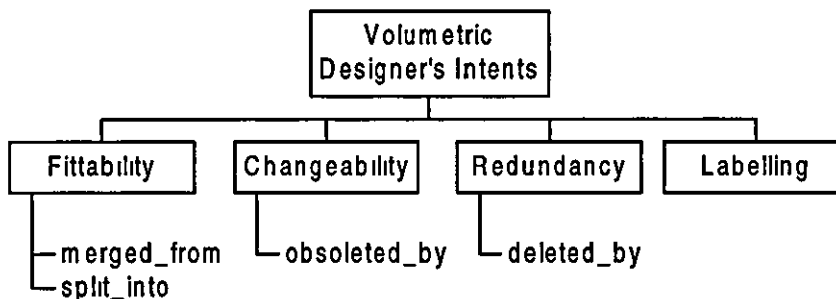


Figure 4-2: Taxonomy of Volumetric FbDI's

4.4.2 PARAMETRIC FBDI

Theoretical Functional FbDI's (TDI's) can be incorporated into feature-based modelling via parametric or variational constraint-based systems respectively generating the so-called procedural and declarative approaches (Shah94a, Shah94b). The main distinction between (constraint-based implementations of) feature-based modelling and parametric/variational modelling has been said to

be in the level of detail/abstraction (Shah94b). Feature models are structures in which constraints can be applied both at microscopic and macroscopic levels.

Validation systems based on the geometric constraining approach do not allow manipulations to be interpreted as a change of the designer's intent, rather they prevent such situations from occurring (Ovcharova94, Shah94b) because the designer's intents are explicitly gathered beforehand via the establishment of a hierarchy/dependency of parameters and constraints. The designer needs a complete understanding of the functionality of the part being designed and all the important dimensional relationships involved. Experimentation is done afterwards when all the relationships are set. Different variations of the part can be obtained easily. However, for loosely specified parts this formal approach is not appropriate.

A considerable amount of research has been done on representing and solving geometric constraints (Serrano88, Suzuki90, Chung90c, Nielsen91, Bronsvort93). Because many theoretical functional FbDI's can be captured and represented via a geometric constraining approach, they can also be called Parametric FbDI's (PDI's)

Constraining is usually a parameter-driven (e.g. dimensions, distances) relation between (parts of) features (Dohmen94). "Parameters may be dimensions, but they may also be values without geometric meaning that are used to compute dimensions" (Dohmen94). Geometric constraining is one of the most important and practised way of capturing (parametric) FbDI's because "much of the design process is driven by functional constraints that turn into geometrical ones as the design proceeds" (Suzuki90). These constraints are important for representing designer's intent as product models because they represent high-level design relationships that must be satisfied and maintained (Emmerik91).

Geometric constraining systems can be roughly divided into two categories: *parametric* or *variational* design (Chung90c).

In a parametric constraining design the designer uses basic geometric elements (e.g. lines, arcs, vertices) and applies a set of geometric constraints (e.g.

parallelism, distance, length, perpendicularism) between the geometric elements. Dependent and independent parameters are assigned and related using engineering equations. A way to relate geometry to function parameters is provided and an equation solver is often available. After completion of the design changes to the dimensions or an equation parameter can be made and all the constraints can still be solved to determine the values of all dimensions of the model. Geometric relationships can only be solved sequentially and equations that depend on the geometric constraints for their solution must be avoided. What the parametric approach can not do is solve the engineering equations and consider the geometric constraints at the same time.

The variational approach allows the engineer to experiment with a design that is under-dimensioned or not fully constrained or specified in any order. In addition, the engineer can specify engineering equations that need to be solved simultaneously with the geometric constraints.

PDI's represent explicitly defined intents that establish tight dependencies between features and feature parameters and thus they are kept distinct from other FbDI's (see criteria in section 4.2).

This work does not contemplate the use of a geometric constraint-based CAD system of any sort (parametric or variational) and therefore these types of FbDI's are not further developed.

4.4.3 GEOMETRIC FbDI'S AND APPLICATION-ORIENTED INTENTS

Although recognised as crucial (Salomons93, Gindy93, Srikantappa94), relational FbDI's have been limited to the separate considerations of *parent-child*, *patterns*, *compound* (Faux86, Emmerik89, Gindy93) and *assembly* relationships (Rimscha90, Nnaji93). Perhaps this is because designers normally do not explicitly state constraints like perpendicularity, parallelism, etc. (Suzuki90, Silva90), except when using geometric constraint-driven systems.

A taxonomy of relational FbDI's has been established by analysing the plethora of designer's intents found in the literature concerned with the feature-based manufacturing and process planning of prismatic parts.

Relational FbDI's describe physical and/or spatial relationships between features and are divided into two categories:

- application-dependent but mostly geometry-dependent, called Geometric FbDI's (GDI's).
- geometry-dependent but mostly application-oriented, called Application-Oriented Intents (AOI's).

4.4.3.1 GEOMETRIC FbDI's

Despite the fact that "it is almost impossible to pre-define all (geometric) feature relationships" (Gindy93), the importance of GDI's has been recognised by systems that incorporate this spatial information in various ways (Dixon87, Suzuki90, Silva90, Nielsen91, Vancza93, Lenau93, Shah94b).

Figure 4-3 presents a taxonomy of GDI's. It is not intended to be complete but highlights important categories and relationships that are found in previously mentioned feature-based systems.

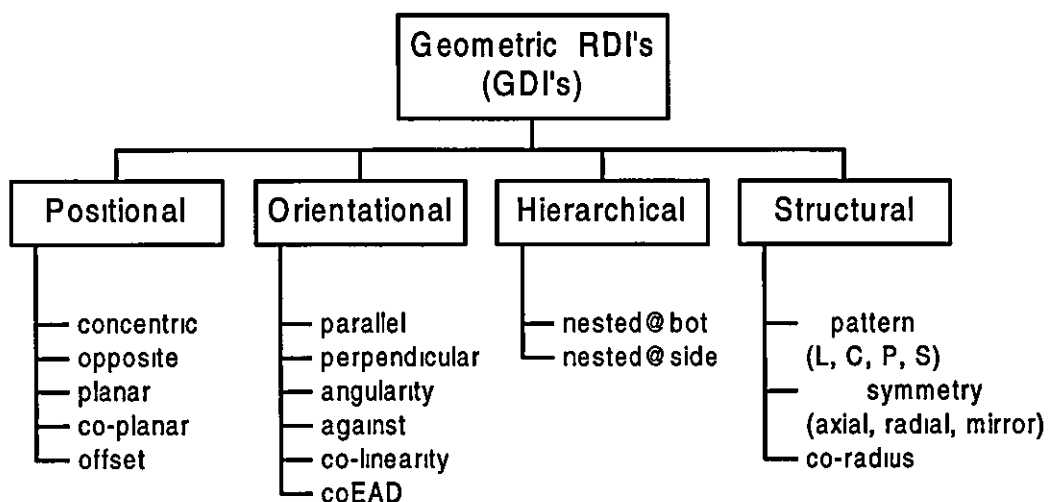
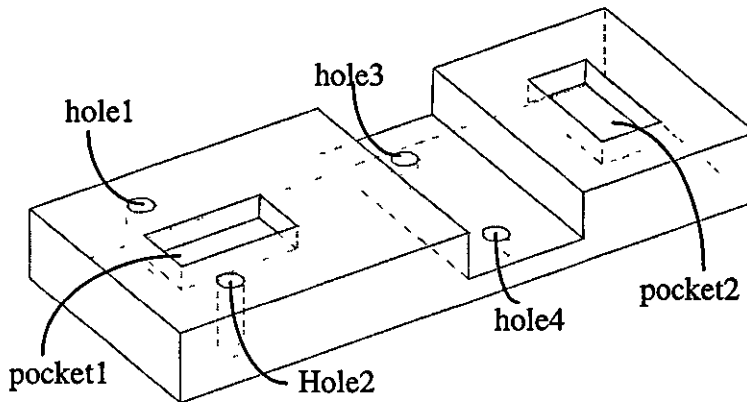


Figure 4-3: A Taxonomy of Geometric RDI's.

4.4.3.1.1 Positional GDI

Positional GDI's are attributes that identify the relative positioning of features that mainly affect feature machining precedence (Mill93, Shah95), process/tool selection and assembly (if assigned between parts, Harun96). They include (Silva90, Chovan91, Vancza93, Srikantappa94):

- *opposite*, if the feature's axes are opposite and co-linear.
- *concentricity* (or *co-axiality*), if the feature's axes are the same.
- *planarity*, if two features have any pair of *virtual* faces that are located inside the borders of one planar face.
- *co-planarity*, if features have *virtual* faces located in the same plane (not necessarily within the borders of the same face).
- *offset*, if both *virtual* faces are located in parallel planes and their normals point in the same direction.



- *pocket1* is *co-planar* to *pocket2*.
- *pocket1* is *planar* to *hole1* and *hole2*.
- *hole1* and *hole2* are *offset* to *hole3* and *hole4*.

Figure 4-4: Examples of Positional GDI's.

Figure 4-4 shows a test part where features manifest various positional GDI's. It should be noted that the *pocket* features share the same plane and thus are *co-*

planar; *pocket1*, *hole1* and *hole2* share the same plane and the same face, and are thus *planar* and; *hole1* and *hole2* share the same normal direction but are in different planes as *hole3* and *hole4*, and are thus *offset*.

4.4.3.1.2 Orientational GDI

Orientational GDI's establish how feature axes or other geometric entities are oriented with respect to each other (Silva90, Rossignac90, Sodhi91, Vancza93, Shah94b, Srikantappa94). Any deviation from these orientational intents during machining operations must conform to a two-bounded quantitative allowance (Parametric FbDI) specified as a *tolerance* (Suzuki90). They include:

- *parallelism*, when the feature axes are parallel.
- *perpendicularism* or *orthogonality*, when the axes are mutually inclined at 90-degrees.
- *against* (or *aligned*), if the features' External Access Direction point in opposite (or same) direction.
- *co-linearity*, if the feature axes are aligned (not necessarily at the same position or in the same direction).
- *angularity*, when an important angle can be identified between the feature axes.
- *coEAD*, when features share the same tool "external access direction" (EAD) usually verified via the orientation of *virtual faces*.

4.4.3.1.3 Hierarchical GDI

Hierarchical GDI's have importance to systems implementation, graphical editing, tooling and process planning. In addition, *assembly* feature relationships are considered to be a structural hierarchy among different parts of a product (Rimscha90, Ovtcharova92). Some hierarchical GDI's can be defined by a *nesting* relationship (Silva90, Anderson90, Dohmen94):

- Nested at the bottom (*nested@bot*) of a feature, when a *virtual* face of a feature is *contained* within a bottom face of another feature (see feature face properties in section 8.4.2).
- Nested at the side (*nested@side*) of a feature, when a *virtual* face of a feature is *contained* within a side face of another feature.

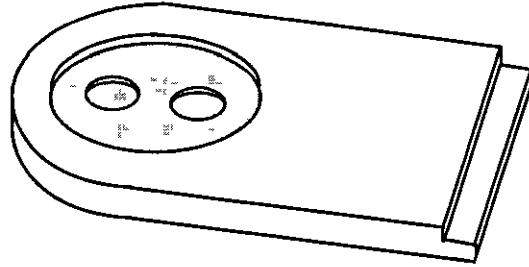


Figure 4-5: Example of *Nested at the Bottom* Hierarchical GDI.

Figure 4-5 presents two blind *hole* features *nested at the bottom* face of another blind *hole* feature as an example of a nesting hierarchical GDI.

4.4.3.1.4 Structural GDI

Structural GDI's recognise combinations of other GDI's such as a linear pattern of features with axial parallelism between each other, and co-axiality of surfaces that have axes perpendicular to a reference. Structural GDI's describe general organisation, placement and orientation of the whole model or of some group of features on the model (similar to *skeletons*, Lenau93).

The *pattern* structural GDI is one of the most popular FbDI's (see Figure 4-6 for an example). Although it has been said that "if not specified in advance it will be impossible to identify pattern features" (Pratt88) it is believed that with the help of the user and guided reasoning some *pattern* structural GDI's can be recognised.

Structural GDI's represent displacement patterns of features (Faux86, Emmerik89, Ovtcharova92) that, for example, affect process planning (Rossignac90, Vancza93) and include:

- *linear (L)*, when elements are regularly spaced along a line.
- *circular (C)*, when elements are regularly spaced on a pitch circle diameter (PCD).
- *planar (P)*, when elements are displaced in a plane forming figures such as squares and triangles (Vancza93).
- *spatial (S)*, when a three-dimensional figure, such as a cube, can be identified as having features at its vertices.

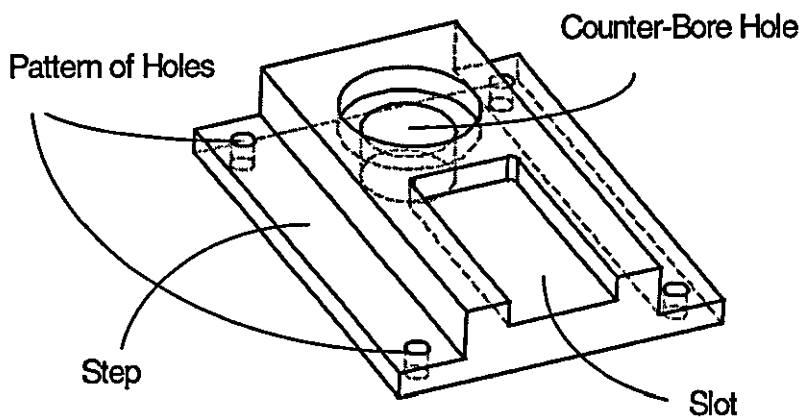


Figure 4-6: Example of a Planar Pattern Structural GDI.

The *symmetry* structural GDI happens when the distance between a feature and a reference is equal to the distance between another feature and the same reference. Symmetry could be assigned to a group of features or to the whole model and can influence the cost of assembly and machine setups (Jakiela89). Types of *symmetry* include:

- *Axial symmetry*, when the reference is one of the Euclidean axes, X, Y or Z.
- *Radial symmetry*, when the reference is, for example, the axis of another feature (say a *hole* feature) (Vancza93).
- *Mirror symmetry*, when the reference is a plane (mirror).

4.4.3.2 APPLICATION-ORIENTED FBDI'S (AOI'S)

GDI's are geometrical facts and intentional relationships between entities in a feature-based modelling system but they alone do not suffice for an application. For instance, a hierarchical GDI is needed in order to define machining precedence but other geometrical reasonings such as "supporting walls" and "tool accessibility" must be considered as well.

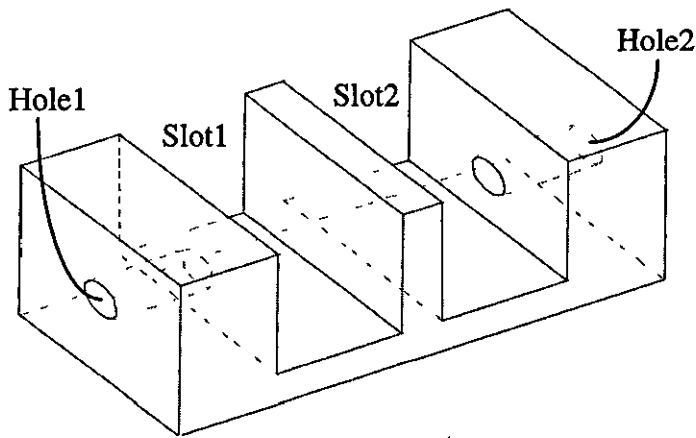
GDI's are defined in the detail geometric designer's domain but there are also "process planning engineer's intents" (Vancza93) as well as "manufacturing engineer's intents", and so on. The intents from all these other application domains are called Application-Oriented RDI's (AOI's). Many of these intents are concerns to be fulfilled that guarantee the physical realisation of the design constrained by pragmatic and technological requirements such as cost, quality, time, accessibility and feasibility.

AOI's exist to establish a more definite interpretation from the application's point-of-view. In contrast to GDI's, these intents consider information beyond geometrical relational facts. This extra information includes tool availability, process optimisation and precedence constraints. Thus different applications could interpret the same factual GDI's differently.

4.4.3.2.1 Temporal AOI

Examples of AOI's include temporal manufacturing relationships. Temporal relationships for machining purposes are embedded in FBM (even if not based on underlying CSG procedural models) due to accessibility of the feature's faces (Hummel89, Gupta93) and are categorised as *coEAD* orientational GDI's. When analysed against other requirements, such as tolerances and time optimisations, temporal AOI can help constrain setup planning (Vancza93, Mill93) suggesting operations to be performed at (see Figure 4-7):

- *same setup*, when features have the same EAD and same feature type;
- *different setup*, when the EAD's do not match.



•*Slot1 same_setup slot2.*

•*Hole1 different_setup hole2.*

Figure 4-7: Example of Temporal AOI's.

4.4.3.2.2 Precedence AOI

A *parent-child* AOI happens when a *nested at the bottom* hierarchical GDI is interpreted as a precedence intent for process planning purposes after an analysis that eliminates other precedence alternatives.

However, other uses of the *parent-child* intent can be found in the literature such as when they are used to model how features are positioned with respect to each other (Laakko93).

Another example of the *parent-child* AOI interpretation occurs when manipulation constraint reasonings are propagated from a parent feature towards child features (Faux86, Shah90, Gindy93, Sheu93). In this context they can be seen as a simplified form of *nesting*, explicitly included by the designer for various reasons.

A final and definite AOI would establish which feature should *precede* the machining of another feature.

4.4.3.2.3 Compound AOI

Another example of an AOI happens when complexly shaped features (and tools) subsume the behaviour of two or more intents and this has been called a *compound* feature (Pratt88, Gao93, Case93c). For example:

- A *counter-bore* or *counter-sink hole* feature, when two *nested at the bottom* hierarchical GDI and *concentric positional GDI hole* features exist;
- Unresolved *Cross-features (X-feat)* such as *stepped-hole* or *slotted-pocket* (Bidarra93, Shah95);
- *Cut-out* arrangement (Mill93), also called *keyway* (Pratt85) or *keyseat* (ElMaraghy93b) which is a “small” recess (another feature) in the periphery of a “larger” feature (Figure 4-8).

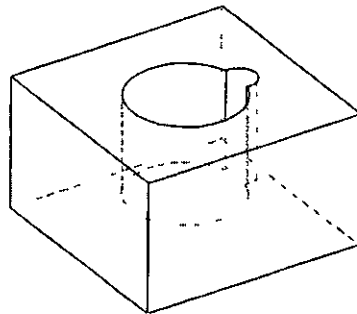


Figure 4-8: A Cut-Out Interaction Case.

Similarly, *T-slots* and *Entered-features (E-feat)* can be defined.

4.4.3.2.4 Proximity AOI

A proximity AOI is usually an intent to be avoided or kept inactive. It is represented by a *thin-wall* (TW) relationship and depends on the material, features and processes involved.

A wall is the unmachined material left between two features. A *thin-wall* can produce a deformation on the component during the machining procedure and therefore should be avoided. Figure 4-9 shows a part containing a *thin-wall* between a filleted *step* and a *slot-through* feature.

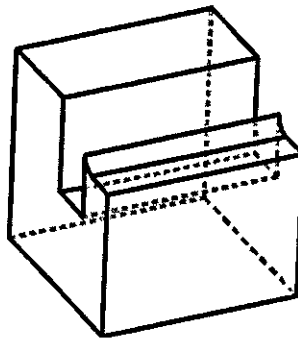


Figure 4-9: A Part Containing a Thin-Wall.

4.5 A FBDI TAXONOMY

Figure 4-10 presents the complete taxonomy of feature-based designer's intents. The leaves of the classification tree are the identified FbDI relationships. It should be noted that because no geometric constraint-driven system has been used, the PDI node has not been further specified (refer to Dohmen96 for a classification of PDI's).

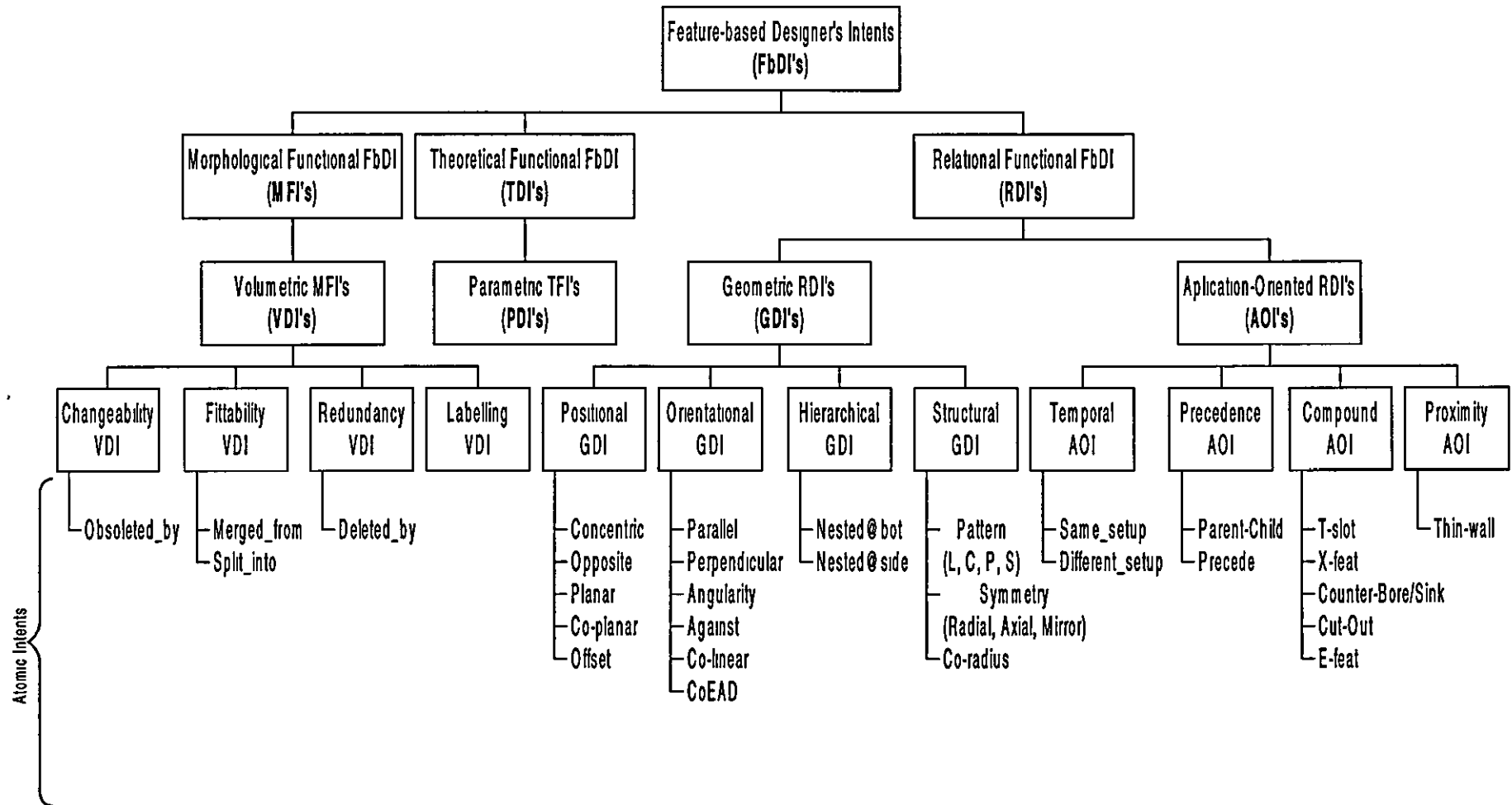


Figure 4-10: A Feature-based Designer's Intent Taxonomy.

4.6 “INTENTURIZATION” VALIDATION

To comply with the *elicitation process* the following are some designer’s intents validation criteria. They help to identify a minimum set of FbDI’s most suitable for an application. Because this has been called featurization validation for features, it will be called *intenturization* validation for FbDI’s:

- Selectable FbDI’s can be conflicting, and hence care should be taken to select only *non-conflicting* FbDI’s for a specific design application. In this way, reasonings will not interfere destructively with each other and loops will be avoided.

For instance, *parallel* and *perpendicular* GDI’s can both be used in the same design or application but they are conflicting if used to relate the same pair of features. Although obvious to humans, the computer model could result in a loop if both intents are assigned to the same pair.

- Because there are partially redundant intents, such as those used to define abstract hierarchical FbDI’s, atomic intents that have *non-overlapping* concepts/definition should be preferred. Thereafter, tricky situations with redundant FbDI’s can also be avoided.

For instance, a *parent-child* AOI could be defined using a *nested@bot* GDI which makes it partially redundant in some implementations. However, these FbDI’s could also have completely dissociated definitions. These definitions need to be clear to the user.

As the intents were mainly gathered from CAD/CAM and CAPP FBM systems, they are consequently all valid candidates for these applications. The intent validation process is relegated to minor importance because no other domain was considered and because no specific application was considered. Therefore, this step of the elicitation process was not fully applied.

4.7 SUMMARY

A better understanding and categorisation of *Feature-based Designer's Intent* (FbDI's) meaning within a feature-based CAD system is a necessary step to foresee how feature reasoning could be embedded into future Intelligent CAD systems. Nonetheless, the feature reasoning process at this level is a complex and difficult matter.

The main objective of this chapter was to distinguish and separate the geometrical, factual and intentional feature-based data from its use and interpretation by an application. In doing so, the complex intent-driven engineering reasoning is reduced to a more atomic level. In addition, because FbDI's are then considered as separate entities, it is easier for an application to store, manipulate and reuse this information. It can also be explicitly and consciously assigned to the design (through a direct instantiation or confirmed automatic recognition).

The taxonomy helps developers to devise feature-based modelling systems that are aware of how FbDI's are captured and represented through the feature concept. It is sufficiently extensive to offer an insight to help obtain a minimal set of designer's intents for a specific implementation via a proper "intenturization" validation.

As features are high-level abstract geometrical entities, they imply multi-level representation and reasoning that should be emphasised for efficiency and expressiveness (Marefat93b). FbDI's presented here help to maintain an abstract and intermediate-level (not as high as the functional conceptual design level and not as low as the geometric detailing design level) vocabulary that feature-based modelling requires and concur to achieve one of the objectives of this research.

5. FEATURE-BASED INTERACTION

Feature interactions are innate to feature-based modelling and pose a difficulty in representation and manipulation for geometric design. This chapter presents a formal and structured geometric spatial feature interaction identification method alongside a broad multi-level classification. Initially, various feature interaction definitions and classifications are surveyed. It was observed that various partial proposals for a feature interaction classification have been made, especially by research involved with the feature recognition approach, but without a general framework. The classification herein encompasses existing feature interaction cases found in the literature and defines a singular framework that leads to a general classification structure.

5.1 THE NEED FOR A BROAD COVERAGE

The use of meaningful entities at an intermediate-level of description such as features and Feature-based Designer's Intents (FbDI's) suggests the need for intermediate-level ways to establish the interactions within the Feature-based Modelling (FBM) context (not as low as geometrical interactions, such as intersection between a plane face and another, and not as high as abstract interactions such as a keyway preventing a cylindrical inlet from rotating). Feature interactions occur when features cannot be considered in isolation

within the model because some meaningful and significant influence is exerted among them (e.g. for manufacturing engineering purposes).

A lack of attention to formalising the concept and classification of feature interaction can be seen in currently available design-by-features (DbF) systems even though this is a well-known, important and active issue of research. Interactions between features are at the heart of any FBM environment because “they are directly and inevitably produced while manipulating the model” (Bidarra93). Besides, *intended interferences* are common practice in engineering and can be found for example in tolerances, assembly relationships or when assigning distribution patterns of features (Zhang93).

Feature interactions have been studied (Vancza93) with particular interest in their effects on the planning capabilities of a CAPP FBM system. A test part containing various identified feature interaction cases has even been suggested as a resource to analyse how a CAPP FBM system would cope with them because, “feature interactions are the cause of some of the most serious problems in the development of generative computer-aided process planning systems” (Mill93, see also section 9.11).

Feature interaction is a major obstacle for DbF systems and some systems simply disallow any combination that potentially creates a physically unrealisable feature. If combinations are allowed then appropriate feature-recognition (FeR) functions are required (Lim95).

Feature interactions are important for determining process sequences and sometimes the processes themselves (Anderson90, Vancza93, Tseng94, Regli96) and it has been claimed that the study of feature interactions is especially useful for feature validation (Allada95).

Therefore, feature interaction urgently needs to be further investigated, precisely defined and established with a comprehensive coverage. This chapter presents a formal and comprehensive feature interaction identification and classification methodology. It aims to apply an *entity elicitation* process and thus obtain a *classification* and *taxonomy* of feature interactions.

A common-sense feature interaction definition is established and a differentiation between this term and related ones found in the literature within the prismatic DbF systems context is presented. Also, the classification and taxonomy presented here will help differentiate feature interaction cases from their corresponding interpretation and use by an application, which has created some confusion with the FbDI concept. Feature interaction classifications are surveyed and placed into a simple and unified framework that is independent and unbiased in its interpretation and application. The classification framework is also an identification methodology based on Boolean operators. It is then shown that this framework can be consistently applied at various levels of the feature-based representation to produce a hierarchical classification for all feature interaction cases.

5.2 TERMINOLOGY

Allada and Amand (1995) distinguished between feature (spatial) *relations* and *interactions*. The former were argued to be non-overlapping situations while the latter alter the feature's internal (volumetric and surface) geometric representation. However, more often both *relation* and *interaction* classifications present *touching* and/or *adjacency* and are based on geometric reasoning. This indicates that these two classifications could be based on the same reasoning and within a unified framework.

For the purpose of this research, feature *interaction* is defined as *a mutual action or influence that exists between features* (Collins87). This definition stresses that an interaction occurs when features cannot be considered isolated within the model and thus, could occur between volumetric overlapping features as well as between non-overlapping and even non-contacting features. Both interaction cases have importance for engineering the component. For instance, features that are volumetrically separate may influence each other due to a *proximity* AOI. On the other hand, if two feature volumes intersect their *morphological* functional FbDI's could have been compromised.

Nevertheless, some confusion exists because the term *interference* is also used in the literature to mean interaction, although it is frequently associated merely with the volumetric overlapping cases of feature interaction. *Interference* has sometimes been used to refer to interactions as a whole because it represents one example of a very important feature interaction case with direct impact on manufacturing decisions. Some authors, for example, would claim that interaction implies intersection between the entities of a feature (Shah90, Silva90, Bidarra93, Tseng94, Martino94a, Gadh95b, Suh95b).

It is understood here that, in fact, *interferences* are special cases of feature interactions where destructive influences occur and possibly lead to a redundancy or loss of the initial properties of a feature or its associated *volumetric intentions*.

The terms *interrelation* and *relationship* have also been used in the literature to actually mean what is here considered to be special cases of feature interaction. Therefore, some confusion can arise between FbDI relationships and feature interaction relationships which the following sections help clarify.

The common-sense definition of the feature interaction term adopted in this research, and defined above, encompasses all special cases above and this definition should be regarded as such hereafter. Geometric spatial feature interaction is based on geometric reasoning for determination and uses volumes, surfaces, dimensions and parameters.

5.3 RELATED WORK ON FEATURE INTERACTION

Feature interaction is an active and important issue but has principally been explored by researchers who are involved with FeR systems. It is considered as a challenge as it has been claimed that the number of features may be finite but features resulting from their interactions are infinite (Kumara94, Tseng94, Allada95, Gadh95b). A consequence is that no general approach to recognise all interactions is yet known (Tseng94, Allada95).

It could be useful to both DbF and FeR systems to identify interaction cases before executing them to produce a set of new features. In this way, extra information is captured that might be significant for subsequent operations and reasoning.

5.3.1 TYPES OF FEATURE INTERACTIONS

Feature interactions in the literature contemplate many different aspects such as:

- *Feature-to-stock* interactions, which happen when the effects of adding a feature onto an existing part's body are analysed (Zhang93, Perng97b).
- *Feature-to-manufacturing constraint* interactions, which happen when features are analysed against their manufacturing constraints such as those interactions between form features and fixtures, datums, (one or multiple) setups or tolerances (Hayes89, Young93).
- *Feature-to-feature* interactions among features from *different representation spaces*, which happen when the interaction is analysed depending on the meaning and effects of a feature perceived from different modelling viewpoints. For instance, interaction analysis between form features and machining features, or functional features and mouldability features (Lee94). These interactions support the analysis and trade-off negotiation process between different feature-based perspectives of the model.
- *Life-cycle* interactions, which happen in a multi-purpose FBM system between features in different application areas of the product's life-cycle (Regli96). This includes relationships between features at the plan-level (mostly within individual components) such as precedence, accessibility and tolerance constraints; at the production-level (on multiple-components or across multiple manufacturing processes) such as scheduling constraints; and throughout the product's life-cycle such as when

manufacturing constraints influence features that impact the part's maintainability and disposal.

- *Explicit* or *implicit* feature-to-feature interactions within the same representation space. Explicit interactions are established categorically by the designer (e.g. geometric tolerances) while implicit interactions are calculated by the system (e.g. "obstruction" and "proximity", Mill93). Implicit interactions are considered difficult as they lack a universal definition.
- *Spatial feature-to-feature* interactions on one component within the *same representation space*. These identify how features are spatially related to other surrounding features in the same part (partially considered by Regli96). Many other authors have proposed sub-sets of feature interaction for this domain, but with emphasis on the impact on manufacturing applications (Pratt88, Gomes91, Su94).
- *Spatial feature-to-feature* interactions of features *among various components*, which specify mating conditions for assembly purposes (also called *assembly features*).

5.3.2 SOME FEATURE INTERACTION CLASSIFICATIONS

Some feature interaction classification proposals are briefly presented followed by a discussion of their drawbacks. It should be noted that the use of different terms for feature interaction are shown underlined

As features have a *volumetric intention* (see section 3.5.2), feature *volumes* (FV's) have been used as the means by which feature interactions are determined. For instance, Su and colleagues (Su94) have attempted to solve feature interaction problems which were classified into two types of volumetric interference:

- *overlap* which happens when there is a non-null volume intersection;

- *combination* (also called *compound* features by Pratt88), which happens when features can be decomposed into a collection of simple features.

Alternatively, a feature interaction classification based exclusively on a feature's *nature* (its additive or subtractive character) has been proposed and used as an important element of a solid modelling scheme for representing features (Gomes91, Kraker95) and was considered the most promising, powerful and yet simple basis for a sound definition of feature interactions (Bidarra96).

Regli and Pratt (1996) divided *spatial* feature interactions into *interference*, *adjacency* and *remote* interactions. However, no formal identification procedure was given, a mixture of geometrical elements was found and the examples did not help understanding: *interference* interactions were defined as having "some volume shared by the two features", but "gripping features" sharing "some common area" were given as an example.

Pure boundary implementations of features have also been used as the foundation for feature interaction classifications. For instance, a fast interaction identification and classification method based on polyhedral features was devised by Talwar and Manoochehri (1994), but was dependent on the internal B-rep scheme and had separate approaches for concave and convex features.

Some interaction classifications exclusively use feature faces (also called "sub-features" or "primitive features", Anderson90) as an internal representation technique (Silva90, Srikantappa94) as well as an aid for feature recognition (Tseng94, Kumara94, Shah94b) and editing (Kim93, Suh95b). Some interactions are between faces of different features and others between faces within the same feature.

Anderson and Chang (1990) considered that there are two critical feature (spatial) relationship types for CAPP applications:

- *Nesting*, classified as *at the bottom* and *at the side* (Figure 5-1), which happens when a contact-type relationship between two features exists and one feature opens into another. Nesting is a special case of a touching spatial feature interaction relationship when one feature's face is *real* and the other feature's face is *virtual* so that the former is contained within the latter.
- *Intersection*, which was classified as between features of "the same type" and of "different types" because of manufacturing importance (Figure 5-3).

Nesting is concerned with situations of the touching of features' faces while intersection is concerned with the type of the features involved (and implicitly with their *volumetric intentions*). This mixture of geometric elements of different types and dimensionality (features and faces) within the same classification framework emphasises that many applications require the classification to contemplate different levels of interactions. Only binary geometric relationships were considered and non-contacting interactions were ignored.

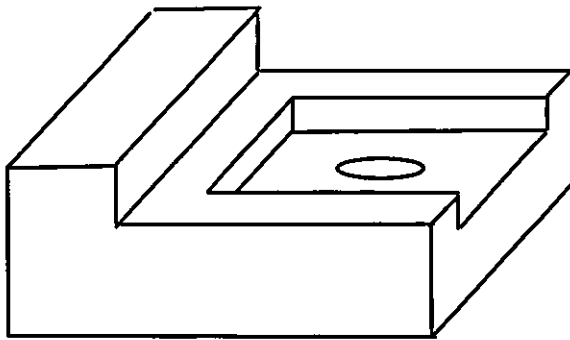


Figure 5-1: Anderson's *nesting at the bottom* Interaction Case.

It can be inferred that nesting interactions are, in fact, a process planning interpretation of a more geometrically-driven relationship (touching) helped by extra information (face property) and reasoning. Thus, there is an associated confusion between the interaction case and its use.

Figure 5-1 shows a feature-based part where a *hole* feature is *nested at the bottom* of a *slot* feature which is also *nested at the bottom* of a *step* feature.

Zhang and ElMaraghy (1993) classified interferences into two categories:

- *Collision*, when a machining feature *volume* intersects the part generating non-functional features, unwanted geometry or non-standard topology (Figure 5-3).
- *Cover*, where even though there is no common volume between the part and the feature, an interference may occur (such as when a protrusion covers a depression feature partly or completely) thus generating inaccessible covered features (Figure 5-5).

Zhang also used a complementary set of criteria at the face level for checking the validity of an operation and also claimed that an interference could be valid in one application but invalid in others.

Similarly, Shah's (1990) classification is based on the effects that the feature interaction phenomena could have on the model:

- a feature could be made non-functional.
- feature (standard generic shapes) could generate non-generic shapes.
- a feature could have its parameters rendered obsolete (see Figure 5-8).
- non-standard topologies could arise (see Figure 5-3).
- a feature could be deleted by another (Figure 5-6).

Bidarra and colleagues (Bidarra93, Bidarra94, Bidarra96) have presented a taxonomy for the feature interaction phenomenon. The classification is based on the functional (topological and geometrical) and technological meaning of the interaction:

- *Topological*. The designer's intent is preserved and an individual feature's parameters maintained, despite the feature *volumes* overlapping. Later work subdivided this into *splitting* (Figure 5-3) and *disconnection* interactions (Figure 5-2).

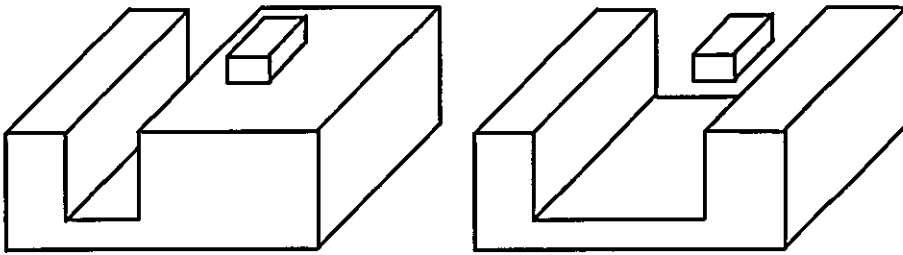


Figure 5-2: Bidarra's *Disconnection* Interaction.

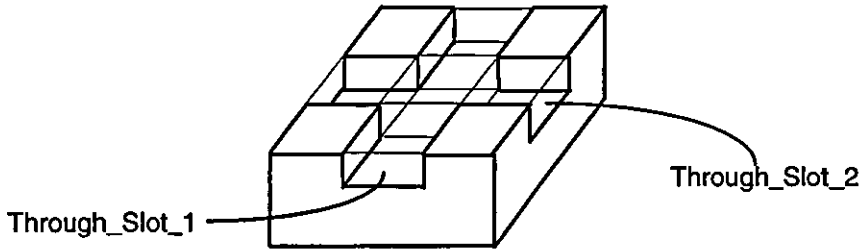


Figure 5-3: Crossing VI Case (Zhang's *Collision*, Bidarra's *Splitting* Interaction, Anderson's "same type" *Intersection*).

- *Clearance*. When a total or partial obstruction of a feature of negative nature occurs (Figure 5-4).

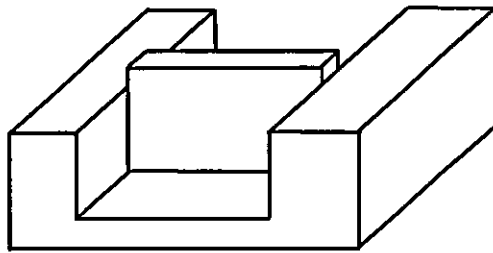


Figure 5-4: Bidarra's Volumetrical *Clearance* Interaction.

- *Closure*. This occurs when access to a feature is closed (Figure 5-5), and can be considered to be an extreme case of the *clearance* interaction in that it causes total inaccessibility of a feature with negative nature.

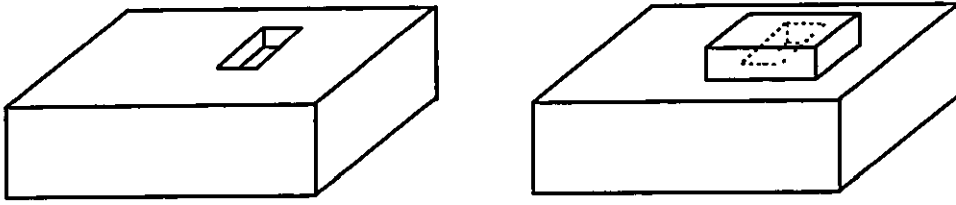


Figure 5-5: Adjunct VI and Inside FI Cases (Zhang's *Cover*, Bidarra's *Closure*, Anderson's "at the bottom" *Nesting Interaction*).

- *Absorption*. The feature's behaviour is absorbed by another feature (see Figure 5-6).

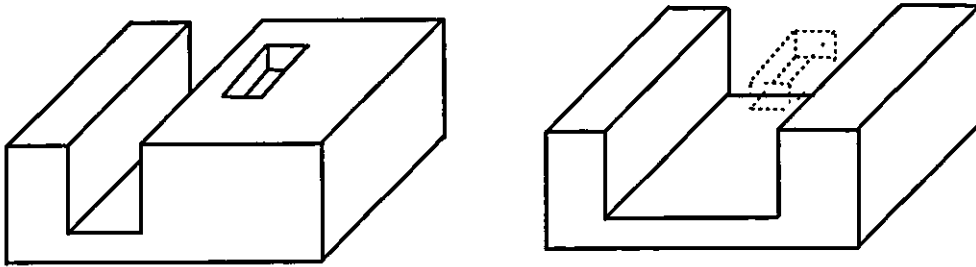


Figure 5-6: Bidarra's *Absorption Interaction*.

- *Transmutation*. The intended semantic behaviour of a feature is destroyed by feature manipulation. For example the insertion of a *slot* may cause encroachment on an adjacent *slot* and give it the behaviour of a *through slot* (Figure 5-7).

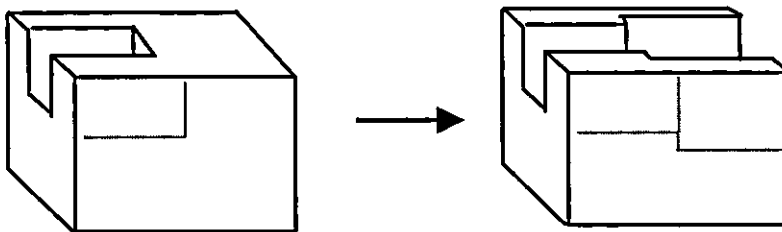


Figure 5-7: Bidarra's *Transmutation Interaction Case*.

- *Geometric*. The feature's geometry is affected without affecting its semantic behaviour (basically parameter-driven manipulations, Figure 5-8). Manipulating one feature's parameters could change another feature's parameters but not its functionality.

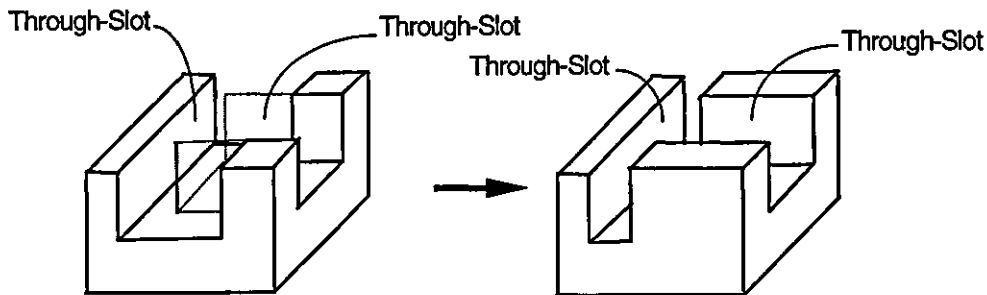


Figure 5-8: Bidarra's *Geometric Interaction Case*.

- *General*. These are interaction cases that do not fit any of the previous cases.

Feature interaction cases and moreover, their associated properties have been used to determine how features can be effectively updated (Perng97b) in a DSG implementation (with its respective B-rep evaluation). Feature interaction cases such as *enclosure* or *intersection* (that happen when part or all of a feature *volume* is removed from the component) were identified. Interaction properties were defined by considering possible interaction scenarios with a third existing feature on the model. Therefore, the editing of a feature would require a possible updating procedure ranging from straightforward addition or deletion of the feature from the representation tree up to a complete recalculation or re-evaluation of both the B-rep and the DSG depending on these properties.

5.3.3 A DISCUSSION ON EXISTING CLASSIFICATIONS

The existing feature interaction classifications, although possibly very efficient in some cases, do not comply with any comprehensive classification scheme, are oriented towards specific applications and are thus, biased and constrained by their domain. Furthermore, “neighbouring” or “adjacency” of features has been considered (Pratt88, Shah90, Lee94) to be of crucial importance for applications (such as computing tool approach directions) but neglected in most classifications because they are not considered to be interferences (Allada95).

Many classifications mix different types of geometrical data during analysis (Anderson90, Zhang93, Talwar94) producing a resulting confusion. For instance, Talwar and Manoocheehri’s classification considered that a feature *contained* by another is in a different class from *intersecting* features, but this contradicts the common-sense understanding of the volumetric intersection operation. This mixture problem suggests the need for a classification framework that could be applied to various levels of geometric information, but in a structured and consistent manner.

Functional technological classifications are prone to have new meanings and types of feature interactions being added (as happened for Bidarra94 and Bidarra96) because this type of classification is dependent on the application’s “understanding” and coverage of the interaction case. Therefore, separating the means of defining the case from its meaning and use would be more appropriate and therefore more application-independent.

It can be inferred that *spatial* feature interactions seem to drive other types of interactions presented in section 5.3.1, and thus should be as accurate, extensive and detailed as possible in order to be used by a great variety of applications. This detailed interaction classification should include various levels such as the volumetric and boundary ones.

The aspects presented above suggest that there is a need to keep interaction identification (calculation and classification) and its semantics (use, reaction or reasoning) as separate processes because of:

- the distinction between *spatial* feature interactions and other types of interactions or relations (Allada95, Regl196);
- the ever growing or functional/technological interpretation (Shah90, Bidarra94, Bidarra96) of the interaction phenomena;
- the widespread use of feature interaction cases to identify application-dependent uses and (Anderson90);
- the fact that different applications could have different interpretations, valid or invalid, for the same feature interaction (Shah90, Zhang93).

The binding of the interaction case to a specific semantic should be a subsequent reasoning dependent on the application so that information concerning the designer, the product, standards with which to comply, manufacturing processes, etc. can be then considered.

From the discussion above, it can be seen that feature interaction classification should:

- consider a broad spectrum, including adjacency and remote cases.
- avoid the mixing of geometric entities but consider all different levels.
- have a unified framework able to be applied to all levels and,
- be independent of the GSM and indifferent to concave and convex features.

This chapter presents a classification framework to identify spatial feature interactions in one component within the same feature representation space and aims to fulfil the above criteria.

The variety of classifications and interpretations presented above not only shows how non-standardised this topic is, but also shows how important and how widespread the application of feature interaction identification and classification is.

5.4 THE CLASSIFICATION FRAMEWORK

The idea is to have a basic classification framework between any two entities (Figure 5-9) and reproduce it at different levels using the same principle and identification procedures. Simple Boolean expressions are used to identify each category.

The entities used in the framework are presented first, details on how the identification procedure works, and the semantics, categories and levels follow and the overall classification structure is presented at the end of the chapter (Figure 5-13).

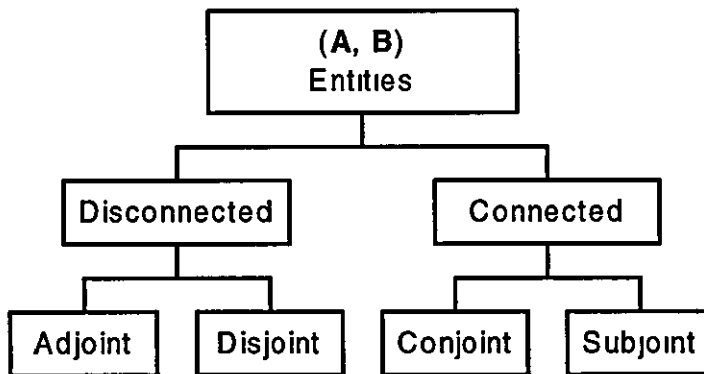


Figure 5-9: The Basic Interaction Classification Framework.

5.4.1 ENTITIES AND LEVELS

The analysis considers a pair of elements at a time, called the *joint* A and B , from a specific entity set (Σ) with a *relative level* (l) . This is denoted by $(A, B) \in \Sigma^l$.

The classification is made according to the results of operations on the *joint*. Table 1 exemplifies entity sets at various levels with their *relative level*. It also exemplifies possible sources of respective entities.

The classification scheme is applied to three levels of interaction: volumetric (VI), boundary (BI) and facial (FI). Similarly to FAV, mentioned earlier, FAB is defined as the Feature's Associated Boundary (closed set of boundaries) and FAS as the Feature's Associated Surfaces (individual faces of the FAB). It has already been shown that many applications need to be able to identify the interaction between features and their components at all of these levels. The framework, however, is consistent and comprehensive for all levels.

Relative level (I) is a term used here only to clarify and to distinguish between entities with respect to their relative complexity and comparative dimensional representation but no mathematical meaning or relationship is used or implied.

Entity Set	Entity Type	Relative Level	Possible Source
Σ^5	FAV	$I = 5$	CSG representation
Σ^4	FAB	$I = 4$	Boundary evaluation of CSG
Σ^3	FAS	$I = 3$	Surfaces of a B-rep.
Σ^2	Edges	$I = 2$	Degenerate Result
Σ^1	Vertices	$I = 1$	Degenerate Result
Σ^0	NULL	$I = 0$	Absence of Result

Table 1: Entity Sets and Examples of Members.

5.4.2 QUERIES TO THE UNDERLYING GSM

Two Boolean operators are used to make enquiries to the geometric solid modeller (GSM): Non-regularised Boolean intersection (represented as \cap) and regularised Boolean intersection (represented as \cap^*). Boolean intersection operations are commonly available in GSM CAD systems and can be applied to volumes, closed boundaries or even faces. These operators are used to obtain **C** and **D** which are the respective results of the intersection operations

on **A** and **B** for a particular Σ^1 . Thus, the operations performed by the GSM and the results used to classify and sub-classify interactions are:

- $C = A \cap^* B, (A, B) \in \Sigma^1$
- $D = A \cap B, (A, B) \in \Sigma^1$

Other queries are the *set membership* tests such as:

- “which feature does the face **F** belong to?”,
- “is the entity **X** of the same type as entity **Y**?”,
- “what is the entity **W**? (a volume, face, edge or, a vertex)”.

This information can be obtained directly from the FBM database because it is usually kept as reverse reference pointers from the FBM to entities in the GSM data-structure.

5.4.3 THE IDENTIFICATION PROCESS

5.4.3.1 CONNECTED OR DISCONNECTED ?

According to the result **C**, interacting entities can be classified into two types: connected and disconnected.

- **Connected** interacting cases occur when **C** is not **NULL**. The word “connected” is chosen to emphasise that the connection between entities will only occur if an entity of the same *relative level* as the inputs is used to establish the connection (and the same can be said of the regularised Boolean intersection).
- **Disconnected** entities occur when **C** is **NULL**, which means that there is no relationship entity of the same *relative level* as a connection between **A** and **B**.

Connected and disconnected are sub-classified by analysing the geometric result **D**, as described in the following sections.

5.4.3.2 SUB-CLASSIFICATION OF CONNECTED ENTITIES

Connected entities are sub-classified into *conjoint* (coincident) and *subjoint* (overlapping) interactions.

Conjoint connected cases are those where one entity is completely superimposed or inserted into another because the output of the Boolean operation is one of the original entities ($C = A$ or $C = B$). Conjoint interaction occurs because the output coincides with one or both inputs. Conjoint cases can be further divided into:

- Cases where the inputs **A** and **B** exactly *match* each other ($C = A$ and $C = B$, which means that **A** and **B** are the same).
- Where one entity is completely *inside* the other ($C = A$ or $C = B$ but, $A \neq B$ or simply, if they are conjoint connected but do not *match*).

Subjoint connected cases (the prefix “sub” when added to nouns refers to an entity, **C**, that is part of a larger one, **A** or **B**, and, in this case, of the same *relative level*), also called overlapping, occur when complex non-standard topologies arise. Such interaction could not affect the entity meaning itself but could have a severe impact on downstream applications.

For instance, if subjoint connected features (Figure 5-3) are not identified and represented properly they will result in redundant machining operations if they have the same *nature*. Subjoint connected cases can be sub-classified into:

- *Enter*, when one entity’s *end* is completely inserted into another entity and a projection of that feature face is inserted on the face it is being projected onto (see Figure 5-10). An entity’s *end* is of lower *relative level* than the entity itself. For instance, a feature’s *end* ($l=5$) is a face ($l=3$), in a similar way that an edge’s *end* ($l=2$) is a vertex ($l=1$).

- *Cross*, when neither *end* of an entity is inside the other (at the same *relative level*) and the *ends* have projections on different sides of the other feature and is inserted on the face it is being projected onto.

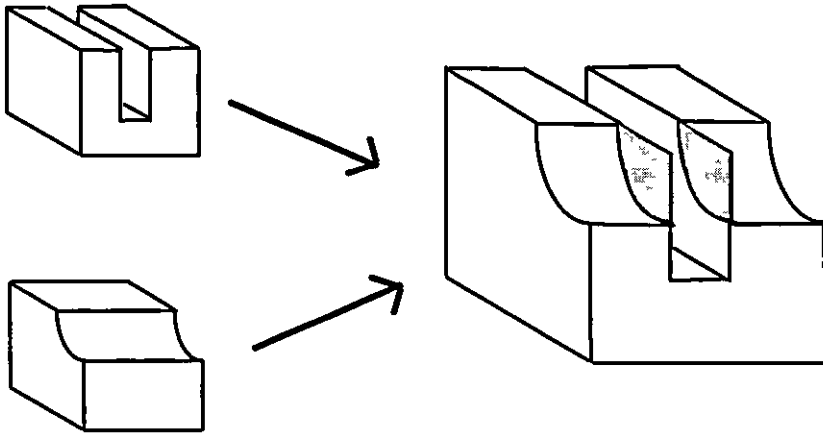


Figure 5-10: Feature's End Face and its Projection.

Figure 5-10 shows two features, a *step* and a *through slot*, combined together in a way to result in the step crossing the *through slot*, regardless of which feature is added first. The shaded faces are the projection of the *end* faces of the *step* onto faces of the *through slot*.

- A range of other cases that can be identified for pragmatic purposes but are left here as a *general* sub-class for simplicity.

5.4.3.3 SUB-CLASSIFICATION OF DISCONNECTED ENTITIES

Disconnected interacting cases (partially considered by Shah90) occur when **C**, the regularised Boolean intersection result, is **NULL**. Additionally, **D** happens to be an entity of an inferior *relative level*. Two situations can occur: *adjoint* (adjacent) and *disjoint* (separate) disconnected interaction.

Disjoint disconnected interaction (the prefix “dis” usually describes the opposite state of something, in this case, the *joint*) occurs when there is no intersection whatsoever, **C** and **D** are **NULL**, and features are considered separate. Disjoint cases are sub-classified as:

- *Far* when the entities are “really” distant from each other (the distance between them is greater than a specified value).

- *Near* when the entities, although not touching, are close to each other and have no other entity in-between.

Conversely, **adjoint** (this word means next to each other, adjacent, touching) **disconnected** cases happen when **D** is not **NULL** and the input entities “share a topological entity” (Shah90, Pratt88) of lower *relative level*, the result **D** (see Figure 5-11).

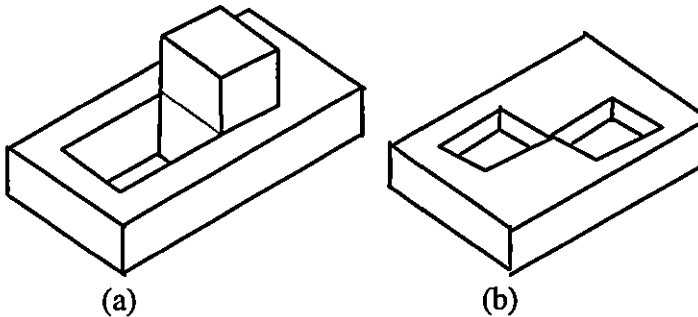


Figure 5-11: Adjoint FI Causing Features to Share a *Face* or an *Edge*.

5.4.4 THE BASIC FRAMEWORK

The framework that uses the entities and procedures presented above is shown in detail in Figure 5-12 and needs to be applied to the three different levels of interest to obtain the complete classification (see section 5.1).

The Boolean operations and *set membership* tests mentioned in section 5.4.2 are reproduced in Figure 5-12 for clarity purposes. Each arc represents a test and each box represents either an operation or a status of the interaction. **A** and **B** are the *joint* entities, **C** and **D** the results of the operations, **m** is the *relative level* of **D** and **n** is the *relative level* of the inputs. The bottom part of Figure 5-12 indicates how this classification framework is related to the levels of interest and presents the few exceptions or special meanings (in brackets).

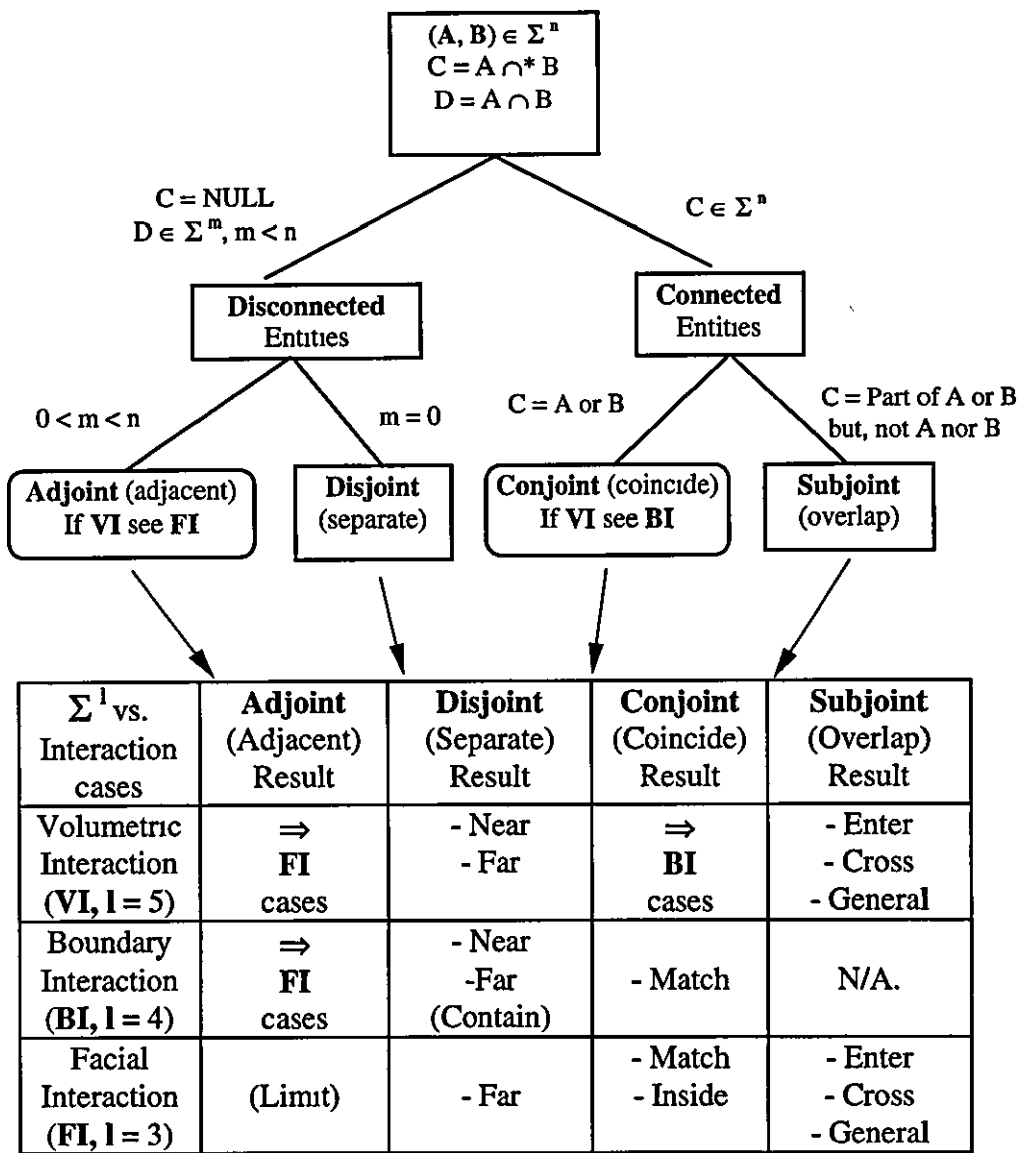


Figure 5-12: The Basic Framework for Classifying Feature Interactions.

The sub-cases that occur at each leaf of the classification tree are also shown in the table at the bottom of Figure 5-12. Some of these are links or pointers to a lower level of the interaction classification. These are identified in the table by the symbol “ \Rightarrow ” pointing to the lower interaction level. The arrows/links indicate that the classification can go deeper (if required) in order to distinguish between cases that otherwise would be treated equally.

5.4.5 THE COMPLETE CLASSIFICATION TREE

To apply the framework to the classification of volumetric interactions (**VI**), the inputs are FV's and the output for a connected VI feature interaction should be a valid GSM volume (solid).

Another way of presenting Figure 5-12 is shown in Figure 5-13 where the following situations should be noted:

- if conjoint connected cases occur at the VI level then the same structure can be applied to obtain further details but at the boundary level (**BI**) which, in its turn, will lead to a FI analysis if an adjoint BI interaction case occurs.
- if adjoint disconnected VI features occur then various interaction cases could be identified with the same organisational structure as the VI cases, but at the face level (**FI**) and these are further detailed in Figure 5-13. Therefore, adjoint VI or BI cases are linked to many FI interaction relationships as required for each face of the feature's realisation.
- the framework in Figure 5-12 is repeated four times at three different levels throughout the taxonomy of interaction cases presented in Figure 5-13.

5.4.6 SPECIAL MEANINGS AND A FEW EXCEPTIONS

A few special meanings and exceptions have been found when applying the framework to the three levels (these are shown in Figure 5-12 as bracketed words and as boxes of discontinuous lines in Figure 5-13):

An adjoint FI case is called a *limit* because it means that one feature is actually being limited by another.

Disjoint BI cases are called *contained* because they identify that one FV is totally inserted into another's FV and they do not touch from the inside.

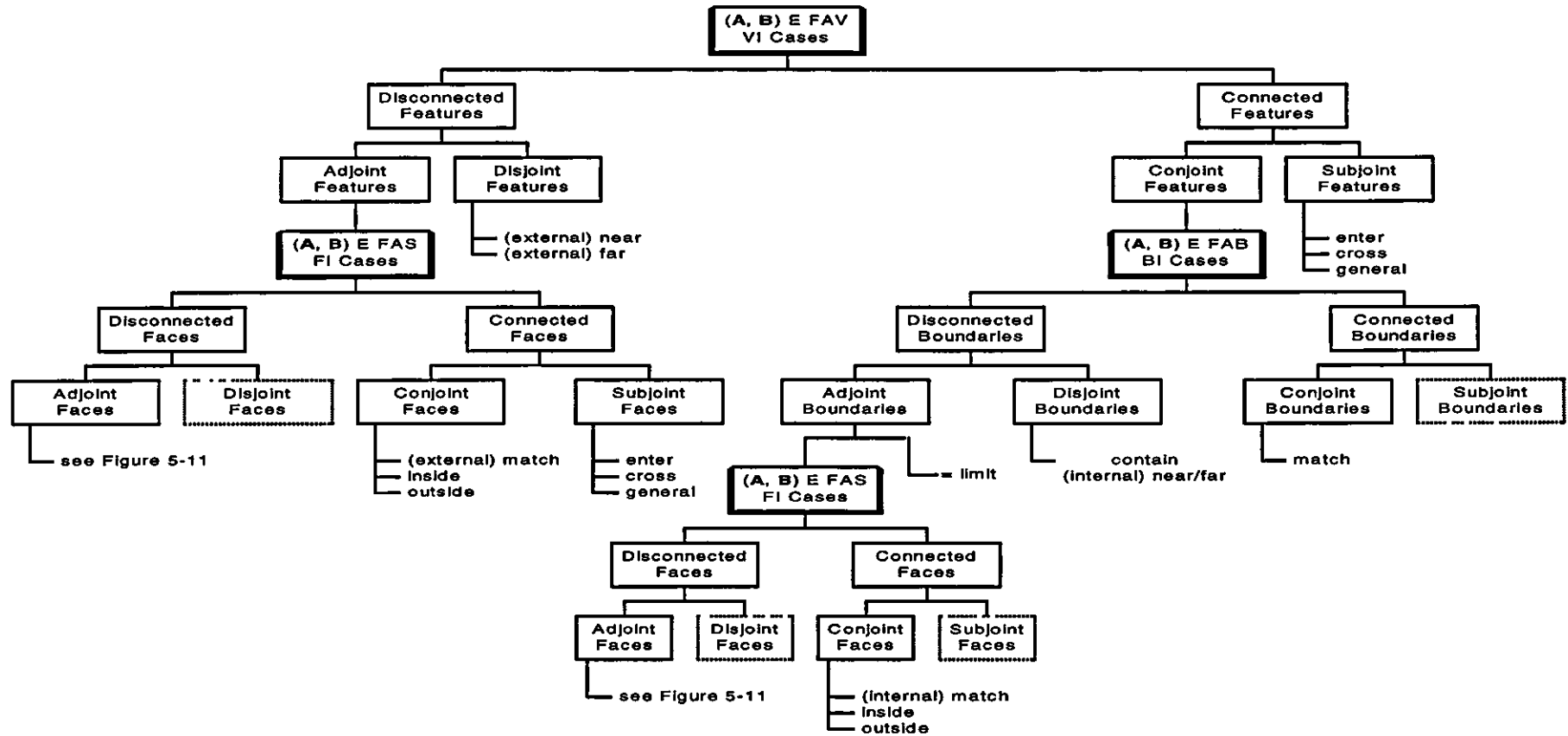


Figure 5-13: A Taxonomy of Feature Interaction Cases.

The interactions presented are not always commutative and thus the interaction relationships have an active or passive response according to which input entity (A or B) is the reference. Hence, active or passive interactions include *crossing* or *crossed*, *inside* or *outside* and *limiting* or *limited*. The exceptions to the active and passive response are the commutative interactions: *match*, *near* and *far*.

BI cases are considered only for conjoint VI cases and this fact affects the meaning of BI sub-cases:

- Disjoint BI cases mean that a feature that is inserted into another's FV does not have external access through the former. It must have an interaction with another feature in order to guarantee that it has "accessibility" to be machined (for example).
- FAB's are considered to be closed set of faces, so no two conjoint VI features would have a subjoint BI case (the intersection operation would return an open boundary) thus, it is marked in the table of Figure 5-12 as Non-Applicable (N/A).

Disjoint FI cases in fact do not happen at all (if derived from BI cases) or as far as this research is concerned do not have useful meaning (if derived from an adjoint VI case).

5.5 SUMMARY

A new feature interaction classification framework has been presented that allows a comprehensive and unified feature interaction taxonomy structure to be conceived.

The classification framework has many advantages such as accuracy (even using Bounding Box data), power (identifies complex cases), elegance (easy to understand), consistency (has a formally defined structure that repeats itself), multi-level (works at volumetric, boundary and face levels) and simplicity (uses simple GSM-based operators and tests). It requires almost no knowledge of the intricacies of GSM representation schemes, although some efficiency is lost because of this.

The classification and identification methodology presented here led to a taxonomy of Feature Interaction cases. It should be stressed that feature interactions have an interpretation in terms of FbDI's that are therefore application-specific but they were presented here separately. Thus, the validation reasoning mechanism will also be responsible for promoting the binding of these two elements.

6. OPERATING FEATURE MODELS

Freedom of manipulation is an intrinsic advantage of using a conventional CAD system and it is taken for granted. For Feature-based CAD systems however, even the most basic manipulation, such as "adding" a feature to a model, is capable of disrupting the validity of a representation. Invalid representations could compromise the usefulness of any analysis subsequently carried out on the model. Thus identifying means to operate feature-based models and the effects that operations have on the validity of the model is a necessity for Feature-based CAD systems.

6.1 OPERATIONS AND VALIDATION

The importance of feature operations as a research issue is that it is closely related to feature validation. In applying feature operators the most challenging task is to handle the interactions between features (Kim96) and the consequent validity of the model.

The importance of feature operations to feature model validation have been stressed by the suggestion of using only "appropriate" valid operations which are responsible for guaranteeing valid output models (Pratt85, Case93c, Zhang93, Kim96). In addition, to *creation* and *deletion*, other operations such as *interrogation* have also been identified as important (Pratt88).

a) The semantics of editing a procedural constraint-driven feature-based model have been studied (Chen95) and editing operations have been classified as:

- inserting or deleting an entire feature;
- changing feature attributes, e.g. from a *blind hole* to a *through hole*;
- modifying dimension values and/or placements;
- changing the dimensioning schema;
- changing the feature shape definition, e.g. changing the cross-section

The first four editing operations were analysed by extracting common procedural steps from which these operations can be composed. This analysis has shown that editing operations of type (1) are different from editing operations of types (2), (3) and (4) (Chen95).

A distinction was made between *generated*, *modifying* and *datum* features. Generated features include extruded or revolved (form) features. Modifying features add chamfers and rounds to edges, or draft angles to a set of faces. Datum features include datum points, axes and planes.

The reported editing problems occur because the procedural evaluation of the model could have, at some stage, a missing reference for its evaluation (a modifying feature can lose the reference to an edge, depending on editing operations on the model). Therefore, an appropriate way of “naming” references was proposed to solve the problem and produce a more predictable behaviour for an edited model. These editing problems seem to originate from the apparent mixture of entities from different levels, i.e. modifying features are, in fact, localised operations on low-level entities such as *edges*, not on high-level entities such as *features*.

- b) Su and associates (Su94, Mayer94) presented a procedure to deal with (validate) feature interaction problems. Operations contemplated by the interaction resolution method include: undo, decompose, reclassify, parameter modification operation, resolution (remove redundant features) and modelling operations (union and difference).

ECTO (Extended CSG Tree of Features, see section 2.7) is the result of the proposed feature recognition process (FeR) where the designer uses union and difference Boolean operations to insert features or modify the parameters of existing features on the part which is subsequently "featurised".

Some phases of this resolution method were executed as background tasks (phases 1 and 3) and others were to be called by the user (phase 2). The operations used on the interaction resolution seem to be unavailable as normal modelling operations and this suggests a distinction between modelling (foreground) and revalidation (background) operations.

- c) Rossignac (1990) studied editing operations of feature-based solid models in terms of the efficiency and representational aspects. The effects of low-level manipulations (such as *face extrusion*) on the syntactic validation of the model were analysed. It was shown that no automated solution exists and that human intervention is necessary to correct the side effects of these editing operations.

To assist the analysis, a rich mixed dimensional vocabulary was defined. It was considered a requirement that "this vocabulary must be 'convenient' so that validity rules can correspond to high-level operations and so that validity rules are simple to formulate and powerful enough to trap common design errors".

It was also claimed that the validity of individual features may not be sufficient to assess the validity of a complex part, and sometimes a relation between several features is also important".

An interesting separation of internal and external operations was made: operations such as editing and even Boolean operations were decomposed into combinations of three fundamental steps (or operations): *subdivision* (which splits the intersection of two objects), *selection* (which associates geometry to features) and *simplification* (which performs deletion or merging without changing the pointsets).

- d) Anderson and Chang (1990) studied geometric reasoning for process planning (such as approach and feed directions, process selection, tool selection and operations sequencing). Features were considered abstractions of manufacturing processes. The operations of *merging* and *splitting* were presented as manufacturing optimisations (e.g. for setup planning).

Splitting was suggested to decompose an unmachinable feature into subfeatures which can be machined separately. *Merging* was said to group features machinable in one fixturing setting. However, a major difficulty in merging was reported in that if features of different types were allowed to merge, the merged feature could be extremely complex and the benefits of merging lost.

It is considered in this research that these two operations are not only closely related to manufacturing analysis but to the concept of features themselves and therefore, both are included for conceptual feature-based representation validation and not for manufacturing analysis.

- e) The two most common operations (*add* and *delete*) were detailed by Zhang and colleagues (Zhang93, ElMaraghy93b) as an attempt to carry out the validation analysis in their system.

An addition operation triggers the following analysis:

- identify if an unwanted interaction has occurred (this includes only *collisions*);

- check if the *connectivity* constraints of the target feature allow it to be related to its *parent* feature;
- test that the feature is not deleting (*covering*) another feature, making it obsolete.

A deletion operation triggers the following analysis:

- identify if features become independent of any parent;
- check if obsolete or non-functional features are produced;
- test if the feature's compatibility with the geometric model was affected;
- investigate if any other inadvertent modification has occurred.

In existing FBM CAD systems the range and number of operators varies greatly depending on the flexibility the system wishes to offer to the designer. These operations have been dictated mainly by the ability to implement a specific operation rather than on an analysis of what operators belong to the designer's vocabulary and therefore should be available in the systems. Furthermore, influences from GSM operations have been observed.

The remaining sections of this chapter describe the application of the *elicitation process* to obtain a classification and taxonomy of operations and a reasonably small set of operations for a DbF conceptual intent-driven validation system. However, *elicitation criteria* are driven only by the proposed classification, *validation criteria* are ignored and the domain is limited to CAD/CAM and CAPP FBM systems.

6.2 OPERATIONS CLASSIFICATION

Three types of operations can be identified: *analysis*, *manipulation* and *setup*.

6.2.1 ANALYSIS OPERATIONS

Analysis operations query about the elements of the model for specific relationships (among features, FbDI's or GSM elements):

- Queries to the FbDI's include enquiries on their *existence* and *status* (active, inactive or dormant).
- The *identification of feature interaction* scenarios (that is all the interactions at a given moment) can be translated into lower-level atomic (Boolean) GSM analysis operations and *set membership* queries.
- Another analysis operation with some interest in the literature (see section 3.1.5 and section 3.1.9) is a *test label* which analyses if a given feature can be verified for all properties associated with a feature of that particular type (i.e. *label*).

6.2.2 MANIPULATION OPERATIONS

Manipulation operations change the representation (implicit and explicit data) in some way. The most frequent manipulation operations are *add* and *delete* but many others can be identified. However, a distinction can be made between *modelling*, *editing* and *revalidation* operations:

- *Modelling* operations are principally responsible for creating the model.
- *Editing* operations are responsible for manipulating and altering the characteristics of an existing feature-based model. They have also been associated with redesigning a part and it has been considered that *modelling* feature-based models is a complicated matter from the

validation point-of-view and *editing* the model complicates it further (Chen95).

- *Revalidation* operations are responsible for manipulating invalid feature-based models (with the objective of converting them to valid models) while the former two operations manipulate already valid representations but potentially leave the model in an invalid state.

6.2.3 SETUP OPERATIONS

The operation of *defining* a (new) feature is neither a query and nor is it a manipulation operation. It is considered to be a *setup* operation (similar to defining the metric system; the dimensional limits of the drawing and the palette of operations to be used in a CAD session). It requires great effort in understanding the chosen system and in programming the new feature (using either a low-level language like “C” or any other high-level language provided by the system).

Similarly, *grouping* features to compose a “macro” feature or for any other purpose is also considered to be a *setup* operation. It should be noted that *compound* features are different from macro features. The former is associated with an application meaning (such as when a *T-slot* is defined due to the availability of a specific tool). The later is an arrangement of, otherwise unrelated, features for manipulation or other purposes.

If a “macro” feature becomes closely associated with an application it should be promoted to a *compound* feature and other characteristics established and programmed (possibly including its automatic recognition by the system) via a *define feature* operation.

6.2.4 THE OPERATIONS CLASSIFICATION

Feature-based operations can be classified into *manipulation*, *analysis* and *setup* operations (see Figure 6-1). As can be seen from the figure, *Analysis* and *Manipulation* operations can be further classified according to the information type involved in the operation. This can be FbDI, GSM or FBM entities.

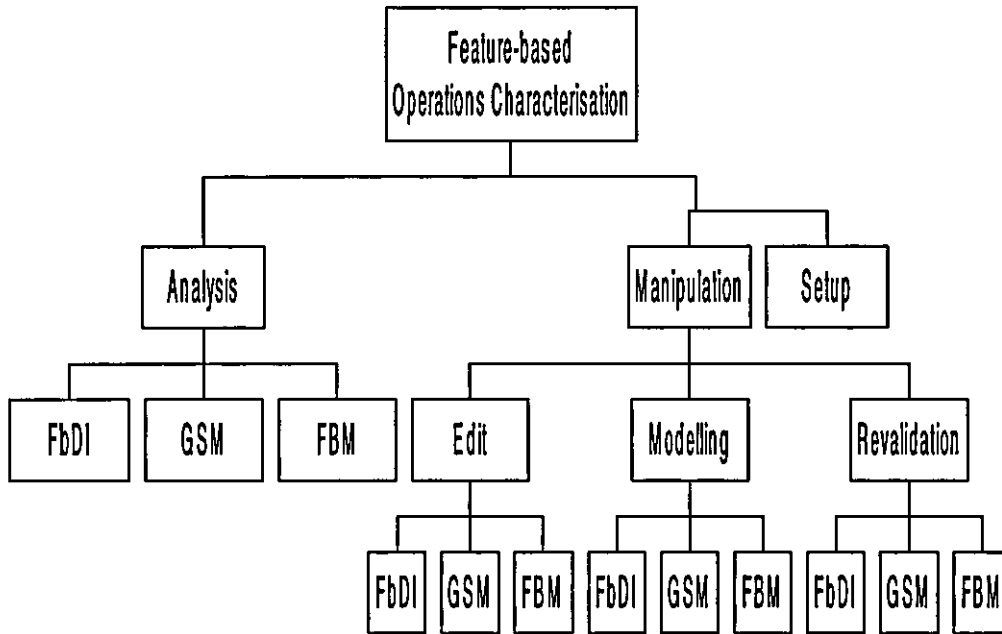


Figure 6-1: The Classification of Feature-based Operations.

6.3 MANIPULATION OPERATIONS

6.3.1 MODELLING OPERATIONS

The following are modelling operations that can be used in a conceptual intent-driven validation DbF system:

- *Add feature*, adds a new feature from a library to the implicit representation database with user-defined parameters and produces a predefined shape on the stock-material. The feature *label* and parameters such as sizes, location and orientation have to be specified.

- If a *parent-child* AOI is part of the hierarchical feature-based representation, a *select parent* feature operation is associated with the *add feature* operation and is performed alongside the add feature operation.
- As an alternative to defining feature parameters, a *derive parameter* operation can be carried out, once a *parent* feature is selected. Similarly, an operation for the attachment of properties or attribute values can be performed (Pratt88).
- *Delete feature*, removes a feature from the feature-based implicit representation database as well as removing its influence on the stock's *volume* and *surface*.
- *Add* and *delete intent* modelling operations create or remove an intent relationship between features, similar to *add* and *delete feature*. Features as well as FbDI's are maintained in the model as lists of active, inactive and intentional entities (see section 7.5) that help reasoning after later manipulations of the model.

Auxiliary modelling operations that reflect the trial-and-error design approach can also be identified:

- The *Undo* operation, returns the model to the status and configuration before the last (addition or deletion) operation.
- The *Redo* operation, recovers the status of the model after the last (*add* or *delete*) operation.

In addition, combinations of the previous operations are recognised as important. Special attention should be paid to them because, for example, the addition of multiple instances of a feature (e.g. *add array of features* operation) in a specified pattern has an associated structural GDI that should also be included (*add intent*).

6.3.2 EDITING OPERATIONS

6.3.2.1 (HIGH-LEVEL) FBDI/FBM EDITING OPERATIONS

High-level editing operations ease the task of manipulating an existing FBM.

They include operations such as:

- *Copy* and *paste* feature operations that have been implemented as a different way of *adding* a feature (Zhang93).
- The *change feature* operation manipulates feature parameters and has also been called *modify feature*, *shrink (stretch)* feature's width (length) (Perng97a, Martino94a). It could be achieved via deletion and re-evaluation of the feature with modified parameters (Pratt88).
- *Move feature*, which is achieved via a translational and/or rotational transformation, is also considered a complex topic, particularly from the validation point-of-view (Pratt88).

6.3.2.2 (LOW-LEVEL) GSM EDITING OPERATIONS

GSM (low-level) editing operations allow the designer to edit a FBM representation at the GSM level and therefore allows the editing of its constituents such as *points*, *arcs* and *edges*.

Martino and colleagues' work (Martino94a, Martino94b) is an example of the use of both feature-based modelling and geometric solid modelling operations but it was emphasised that the use of the latter can give rise to "degenerated" situations in which the existing features no longer have meaning or some of their characteristics are changed. It was claimed that this could only be corrected through a feature recognition process which not only recognises the (solid) modelling operation in terms of features but also updates the feature model according to the effects of the last (geometric) operation.

Besides the use of GSM Boolean operations, other low-level editing operations include:

- *Change cross-section or (sweeping) profile* (Chen95);
- *Change constituent*, such as *move vertex*, change a *straightedge* into an *arc* or *enlarge a face*;
- *Apply chamfer*, in the form of *add a modifying feature* (Chen95);
- *Apply fillet*, in the form of *make fillet feature* (Perng97b).

6.3.3 REVALIDATION OPERATIONS

In earlier chapters it was made clear that verification is not only an important task, but that it is of equal importance and usefulness to be able to operate the model when an invalid situation is found. This gives rise to *revalidation* manipulation operations.

Revalidation operations can be applied *automatically*, but are usually applied in an *assisted* manner. Editing operations can produce substantial topological changes that require user intervention (Chen95). Revalidation operations have even been proposed (Stroud93) as a supposedly general strategy for handling all types of information in a product model (including B-rep and feature models) to maintain the integrity of the data structure. However, it was still found necessary to “request a user to verify if the information is still correct” after an operation.

Although *delete* operations can cause a *wound* in the model and therefore a *wound healing* strategy should be devised to maintain the validity of the model (Zhang93), it is believed that all manipulation operations, and not only *delete* operations, require subsequent revalidation operations.

If modelling and editing manipulation operations are defined without the assurance of a valid result then revalidation operations need to be identified as a separate set of operations that can be used on request but would preferably be applied automatically.

If, on the other hand, one implements validation via a set of “valid” editing and modelling operations that by themselves guarantee a valid result, then revalidation operations can be associated with, and indeed embedded within, these operations. Nevertheless, even for this last case it is possible and beneficial (e.g. for implementation or optimisation reasons) to identify and isolate revalidation operations.

Revalidation operations are listed here regardless of possible previous operations. It is recognised though that previous operations can be used as clues to a better way of dealing with invalid models subsequently generated. The following are the atomic revalidation operations identified:

- *Add volumetric intention.* This revalidation operation is similar to the *add feature* modelling manipulation, but manipulates FV’s that will be later identified as a feature (via a proper *search label* operation). *Add volumetric intention* is usually requested after other revalidation operations such as *split*.
- *Delete volumetric intention.* This is similar to the *delete feature* modelling manipulation, but occurs when FV’s are of conflicting *natures* and are therefore inactivated. An *add intent (VDI) deleted_by* operation is required when the *delete volumetric intention* revalidation operation is performed.
- *Make feature obsolete.* When a feature’s *volume* has a conjoint VI interaction (overlaps completely) with another feature’s *volume* of the same *nature*, then this latter feature is said to have been “obsoleted by” the former and thus it is removed from the model but is kept in a dormant or *intentional* status. An *add intent (VDI) obsoleted_by* operation is required when the *make feature obsolete* revalidation operation is performed.
- *Activate feature.* Features that were made *intentional* or inactive in the model can become part of the active model again via this revalidation operation. The situation that originally caused the dormant feature should have been resolved otherwise a possible loop would arise. For instance, obsoleted (deleted) features can become active and reappear in the model if

the overlapping (deleting) feature is later removed. A *delete intent* (VDI) *obsoleted_by* (*deleted_by*) operation is required when the *activate feature* revalidation operation is performed.

- *Split*. Divides the FV of a feature against the FV of another one, usually producing two or three new “smaller” FV’s using convex FV’s. The initial FV is *deleted* and the resulting FV’s must be *labelled*. An *add* (VDI) *intent split_into* operation should be applied between the inactive FV and the newly generated features. This revalidation operation helps correct obsolete parameters of the feature.
- *Merge*. Combines the *volumetric intentions* (FV + FN) of two distinct and adjoint VI (touching) features producing a larger feature that needs to be properly *labelled*. The initial volumetric intentions are *deleted* and the merged feature is added to the model. An *add* (VDI) *intent merged_from* operation should be created between the inactive FV’s and the newly merged feature.
- *Labelling* is responsible for operating on the feature’s parameters at the face level and finding a proper meaning for the result – the feature’s *label*.

Section 3.5.2 presented the *labelling* VDI and features as being represented through a template of face properties (*real* and *virtual* faces) - see Figure 3-9 and Figure 8-19. A proper feature *label* is obtained by comparing the actual status of all feature faces and the available templates in all possible orientations. The importance of distinguishing and reasoning with types of feature faces has long been considered to have great potential for validity analysis (Pratt88). If a feature’s face property changes, the new configuration can be compared against the template and the *labelling* VDI therefore achieved.

Dealing with features via template definition, and supposing that identifying face properties is a fast and easy task, could give rise to a localised feature recognition mechanism. Localised FeR considers only a limited amount of information surrounding the modified feature to perform

its task and has been receiving a lot of attention in recent research (Laakko93, Martino94a, Su94). Stroud93 said that “using feature verification only on modified parts of an object means that feature structure maintenance is faster than performing a global re-recognition of a feature structure”.

The *labelling* revalidation operation consists of three atomic operations:

- Change face’s property to *virtual* (*to_V*).
- Change face’s property to *real* (*to_R*).
- *Search label* on the feature library (find a *label* considering the pattern of *virtual* and *real* face codes) making sure that all possible orientations are tested.

Ultimately, the *search label* operation is responsible for keeping the function-to-shape relationship match of the features in the model, as defined by the template of every feature's type.

- *Complement* is the operation that converts a representation that includes features with a positive *nature* into a representation with features only of negative *nature* capable of producing the same final shape on the part. Features of negative *nature* have special significance for machining purposes. This conversion can be done by “growing” the stock material and then adding negative *nature* features that generate the original shape. The difficulty lies in the multiplicity of alternatives that come from this approach (Li90, Perng90, Chamberlain93, Waco94, Tseng94), but it is important for feature-based modellers that represent features internally in a DSG scheme.
- *Rigid propagation* extends the effects of an editing or modelling operation to another feature. Propagation is an important and valuable revalidation operation that should be carried out or suggested particularly when a

parent-child or a *compound* AOI interaction exists between features such as *counter-bore*, *parent-child* or *T-slot*.

Nevertheless, other GDI's (such as *concentric* and *parallel*) can require such operations. Rigid propagation changes the positioning and orientation of features and is much easier to tackle than any non-rigid geometric transformations (e.g. scaling). Rigid propagation is usually associated with high-level editing operations.

- *General propagation* is required to propagate changes, usually originating from low-level editing operations, towards features that hold an intent relationship with the feature being edited. The difficulty with *general propagation* (and its difference from the *rigid propagation* revalidation operation) lies in identifying what to propagate.

For instance, a *shrink* low-level edit operation on a feature that has a *nested at the bottom* feature would require the latter to be moved in two or three directions after having the former had been shrunk. Sometimes, another low-level edit operation is required and therefore these operations require the designer to fully understand the object and are difficult to automate.

- *Add* and *delete intent* are also used as revalidation operations because some other revalidation operations can be better specified when the addition or deletion of subsidiary intents can be described.
- *Make intent obsolete* is another revalidation operation that, similar to *make feature obsolete*, makes an intent become dormant or *intentional* in the model, possibly because its related features are either dormant themselves or one of them is dormant or inactive.
- A dual operation of this one is the *activate intent* which brings the intent that was dormant back to a list of active FbDI's.

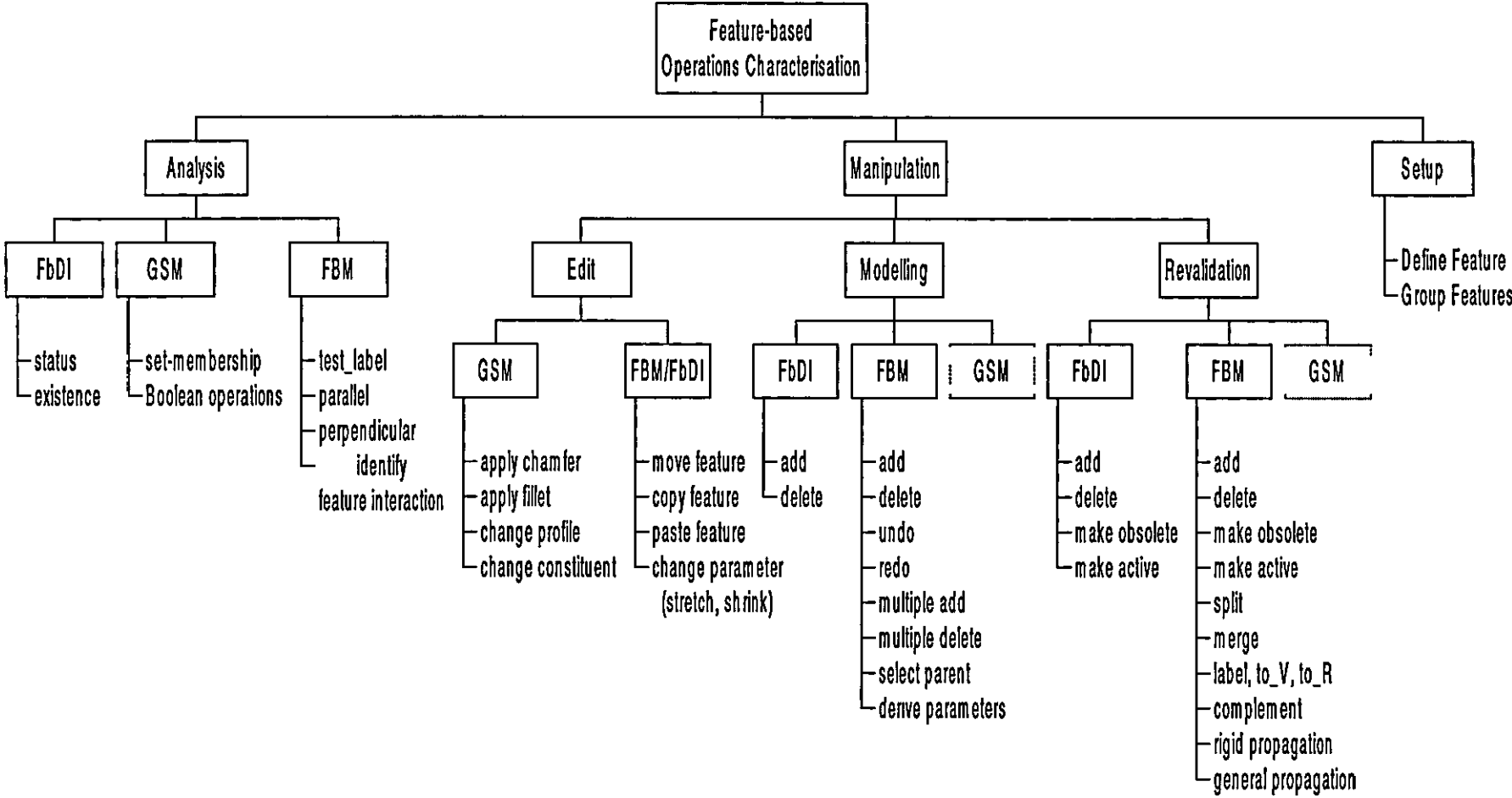


Figure 6-2: The Taxonomy of Feature-based Operations.

The use of internal operations, such as revalidation operations, has to be carefully planned to avoid endless loop iterations. For instance, systems that perform propagation on *parent-child* constraint-based features are prone to a chain reaction problem (ElMaraghy93b). Similarly, internal operations, such as rearranging a feature-based representation, have also been reported to cause endless iteration and thus an *undo* operation must be automatically issued or suggested to the user (Su94).

Figure 6-2 presents the taxonomy of feature-based operations identified in this chapter.

6.4 A MINIMUM SET OF OPERATIONS

High-level editing operations have frequently been implemented as combinations of *add* and *delete* modelling operations because these operations seem to suffice for most implementations (Pratt88, Zhang93, Laakko93, Su94, Kim96). Even a domain-independent formalism for a feature-based design has only formally defined *add* and *delete* feature operations because other operators were said to be defined through these two (Kim96).

For instance, the *change feature* operation is usually implemented as *delete feature* followed by an *add* (a new modified) *feature* operation. It is considered a complex manipulation because *delete* operations *per se* affect children and surrounding features at the same time that these same (child and surrounding) features could have been affected by the addition of the new feature.

The advantage of using *add* and *delete* modelling operations to implement high-level editing operations is that all validity checks and rules defined by the modelling operations can be “inherited” by the editing operations (Zhang93).

Similarly, it is considered that, alongside to add and delete feature, add and delete FbDI comprise a minimum set of modelling manipulation operations in a feature-based intent-driven system.

6.5 SUMMARY

Instead of defining operations *validation criteria* for a particular application, a minimum set of operations have been identified.

Low-level editing operations represent the same manipulation freedom usually found in conventional and GSM CAD systems but they introduce a complexity factor that would require a full implementation of a FeR system inside the DbF implementation in order to cope with the variety of resulting features. Therefore, for most systems low-level editing operations have been disallowed.

7. VALIDITY CONDITIONS

Validity conditions are the central elements in the conceptual feature-based validation framework. They help assess the integrity and consistency of other elements/entities already described. The analysis has been divided into various aspects which generated sets of reasonings. These reasonings comprise the verification statements that guarantee the validity of the model, although this is done via invalidity tests. These reasonings and their organisation are presented in this chapter.

7.1 ORGANISING THE REASONING

Validity Conditions are the medium by which conceptual feature-based representation validation is performed. They are the translation of FbDI's into verification statements. Validity Conditions are also responsible for the binding process of the various elements of a conceptual feature-based validation system: features (and their attributes), feature interactions, feature-based designer's intents (FbDI's) and feature operations. This is done in a way to express feature semantics as a 3D solid modelling technique and relationships with applications. To achieve this, a structural analysis of the reasoning is required.

- a) Priority or sequencing can also be found in other feature-based systems that are said to capture designer's intent (see Dohmen and colleagues' constraint-based systems in section 2.7): different types of constraints are

used in the same design and in the same view and they are maintained through a pre-defined sequence of resolution

- b) Multi-level methods specifically aimed at the validation problem have also been suggested in a study of a formalism for feature-based design validation (Kim96, see section 2.7), which defined *model* validation (divided into *syntax*, *domain*, *feature* and *product* levels) and *functional* validation.
- c) *Local* and *global* validation for ongoing design have also been established (Requicha89b, Rossignac90) in addition to *model* and *functional* validation (Kim96). For instance, accessibility analysis is performed when features are added to the model (*local* accessibility) but also another similar and complementary analysis is necessary (*global* accessibility) which can only be carried out at a later stage when the model is complete.
- d) Phases are another way to organise the validation method and this has already been presented as the resolution of the feature interaction problem (Su94); Phase 1 considered volumetric analysis, Phase 2 considered labelling analysis and Phase 3 performed grouping of unresolved intersecting features into complex (thereafter resolved) feature sets that might have importance for an application.

Given the above examples of how validation reasoning has been organised in related systems it can be assumed that some sort of organisation (sequence, hierarchy or priority) must be devised.

7.2 INVALIDITY TESTS

A human-based analysis of a feature-based model is usually accomplished by searching for invalid situations and therefore much of an engineer's experience is built on the search for invalidity, rather than validity. Although the spectrum of invalid situations is extremely extensive and application-dependent, it has been stated that validity rules must precisely characterise invalid situations (Rossignac90). To facilitate the analysis, invalid situations are divided into subjects that relate to different areas of expertise such as process planning, setup planning and manufacturing. Thus, it becomes easier to identify and devise tests for invalid situations than to identify and test valid ones, especially in the context of abstract elements such as features that have no mathematical or well-accepted definition.

Furthermore, invalidity rather than validity tests are suitable for division into sub-cases that correspond to specific remedies – the revalidation operations. Therefore, it is pragmatically easier to perform feature-based validation on a model representation via invalidity tests.

Nevertheless, from the point-of-view of logic, if a model fails all invalidity tests it can not be considered “completely” valid, but may be thought of as a **non-invalid** model for that specific set of criteria. On the other hand, because no practical distinction can be made between valid and non-invalid models, they are considered similar in this text and both will be called valid models.

7.2.1 THE VALIDATION PROCESS

The following discussion presents validation as the binding process of all elements detailed in previous chapters. In particular, how invalidity tests and revalidation operations work to guarantee a valid model representation output is considered.

The process of validation is an analysis loop (see Figure 7-1):

- A feature-based modelling operation starts the validation process. It can alter the configuration of features, FbDI's and/or feature interactions. FbDI's are verified through invalidity rules that can become active at any time. For a rule to be active all its conditions must be fulfilled making it ready to be executed (fired). A number of rules can become active simultaneously but only one is fired.
- The knowledge-based system selects a rule to be fired according to a priority strategy (Patterson90, Giarratano93) assigned between rules. If the configuration of features has been manipulated then a consequent new feature interaction scenario is calculated (initially at the volumetric interaction -VI- level) and reasoned against all rules.
- Rules perform actions on the feature and FbDI representation via *revalidation* operations (which are intended to simplify the situation on each loop execution). Every time a rule is fired, some active rules can become inactive and vice-versa producing a new set of active rules from which another one is selected to be fired.
- A rule exists which determines when another level of the feature interaction scenario is to be determined. This new scenario is again considered by all rules.
- All features and FbDI's affected by the revalidation operation are automatically considered by the reasoning. A certain degree of unpredictability in execution control is expected concerning which feature and which FbDI will be reasoned first (this is a characteristic of the knowledge-based system implementation, Chung90b, Giarratano93). However, this should not make any difference to the final result.

- When all features and FbDI's are verified and no more new scenarios are produced, the validation process loop delivers the resulting feature-based valid model.

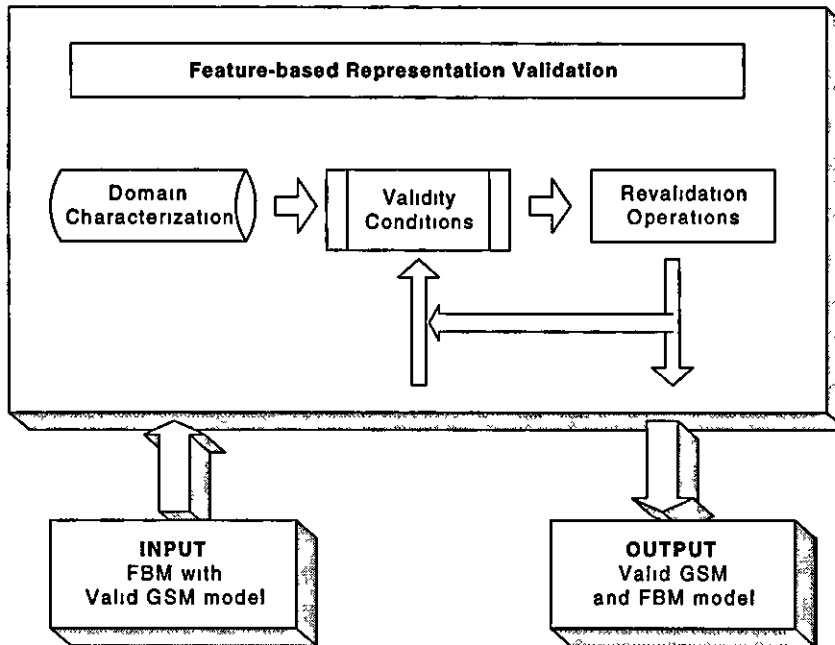


Figure 7-1: Feature Validation Reasoning Framework

The input in Figure 7-1 is a feature-based model just after a modelling operation is performed and is assumed to be an evaluated and valid GSM representation.

The framework shows that the *domain characterisation* (the elicited FbDI's) drive the *validity conditions* (invalidity tests) being considered and therefore the extent of the validation process. The output of the validation process will be both a valid geometric and a valid feature-based model representation. This validated model can then be used by any application and no misrepresentation should exist from the perspectives of the selected FbDI's.

7.3 ORGANISATION

Initially the rules were conceived for *morphological functional* conceptual validation reasoning without any concern for their global organisation because the emphasis was on the usefulness and feasibility of the *conceptual validation* framework. However, the initial prototype implementation revealed an interesting organisation of the relationships, which is described below.

7.3.1 REASONING ASPECTS

Morphological functional FbDI's (MFI's) and the associated *volumetric* FbDI's (VDI's) give a clearer definition of feature semantics. The selective execution of revalidation operations guarantees the delivery of valid representations from the FbDI's perspective.

An analysis of VDI's suggests that there are two aspects that help describe a feature's behaviour: the volumetric interaction aspect and the labelling-dependent aspect.

- Volumetric interaction aspects occur when feature *natures* (FN) are considered and when feature *volumes* (FV) interact at the volumetric (VI) or boundary (BI) levels. The absence of this aspect means that no interaction analysis is considered whatsoever, another geometrical analysis is being performed or a low-level interaction (the face level - FI) is considered.
- Labelling-dependent aspects occur when the feature *label* is the major affected element or when the *labels* of the features involved affect or determine the reasoning.

Volumetric interaction aspects are related to *changeability*, *fittability* and *redundancy* VDI's while labelling-dependent aspects are related to *labelling* VDI's.

7.3.2 REASONING SETS

The combination of these two concerns gives rise to four sets of reasonings according to whether or not they are part of the rule (see Table 2). The volumetric interaction aspect is identified by **V** while the labelling-dependent aspect is identified by **L**.

		Labelling-dependent Aspect (L)	
		Without	With
Volumetric Interaction Aspect (V)	Without	a) - L, - V	c) + L, - V
	With	b) - L, + V	d) + L, + V

Table 2: Sets of Validation Reasoning.

These four sets of reasonings identify distinct and important situations when dealing with *conceptual* feature validation:

- Situations of type (a) are responsible for performing the geometric feature interaction scenario identification as well as any other geometric reasoning. Of particular interest are those geometric reasonings that are simpler, straight-forward or already available in geometric terms than if considering extra feature-related information.

Situations of type (a) perform *simply geometrical* (-L, -V) analysis/reasonings, but do not include GSM validations.

- A situation of type (b) happens when volumetrical reasonings and/or the feature *nature* are enough to fire an action such as when conflicting volumetrical intents (*hollows* or *satellite* volumes) appear in the model.

Situations of type (b) perform *simply volumetrical* (-L, +V) analysis. *Simply volumetrical* tests also include those where an incoming feature interacts with the stock material, regardless the former's *label*. This last reasoning example has priority because the stock material is considered to be the envelope of the whole component (and all its features) and thus, any volumetrical analysis involving the stock would speed up the processing of the newly added feature.

- A situation of type (c) happens when labels are the main focus of the reasoning, such as when the system is searching for the correct *label* for a specific feature according to its face properties.

Situations of type (c) implement *simply labelling* (+L, -V) reasonings. *Simply labelling* reasonings include all those where low level interactions (face level - FI) result in a change of a feature face property (from *virtual* to *real*, or vice-versa) and consequently results in a change of its *labelling* VDI, regardless the feature's *nature*.

- A situation of type (d) happens when both the feature volumetrical interaction and *label* aspects determine the actions to be taken. They are called here *complex* (+L, +V) reasonings. All other reasonings between features, except the stock, are also considered as *complex* verification statements.

7.3.3 PRIORITY

It was found that a priority scheme exists among the four reasoning sets in Table 2 such that every time a situation of higher priority occurs, it is dealt with immediately and then in a descending order of priority up to the point where there is no pending situation. Within the same priority level any sequence of rules can be expected to be fired as previously explained. The priority found, from the highest to the lowest (see right-hand side of Figure 7-2), is:

1. **Simply geometrical reasonings** (type (a): -L, -V).
2. **Simply volumetrical reasonings** (type (b): -L, +V).
3. **Simply labelling reasonings** (type (c): +L, -V).
4. **Complex reasonings** (type (d): +L, +V).

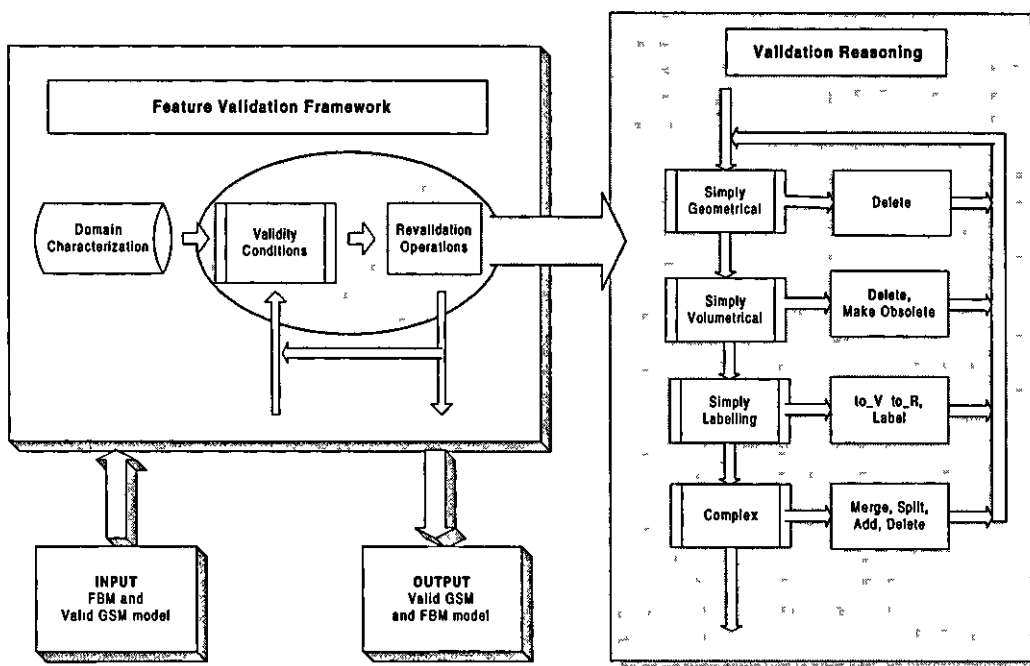


Figure 7-2: Sets of MFI Reasonings.

Simply geometrical reasoning performs GSM reasoning and generates the interaction scenario between features at various levels of interest (initially volumetric interaction up to face interactions), as it is requested.

The feature interaction scenario is then considered by the subsequent sets of reasonings. The first of these is the *simply volumetrical* reasonings. If there is enough information, the *labels* are verified and (re)assigned via *simply labelling* reasonings.

If the model is still not valid then, there will be enough information with both *labels* and feature interactions defined and corrected. In such a case, face interactions are added and *complex reasonings* are then performed.

7.4 INTENTS MANAGEMENT

Conceptual feature-based representation validation was performed via MFI reasoning. MFI reasoning is not only responsible for identifying invalid morphological situations and deploying revalidation operations but also for *adding* and *deleting* VDI relationships. Occasionally, it is possible that RDI relationships may be created by MFI reasonings. This suggests that MFI reasonings drive some RDI reasonings. However, there are RDI reasonings that are independent of MFI and feature interaction cases. In other words, there are RDI's that are dependent on their own functional meanings and therefore have their own reasonings.

An intent-driven paradigm suggests that a feature-based modelling application could reason not only with FbDI's embedded in features (as most of the applications surveyed claim to do), but should also be able to reason about FbDI's themselves. Therefore, means to validate, recognise, manipulate and manage FbDI's are required.

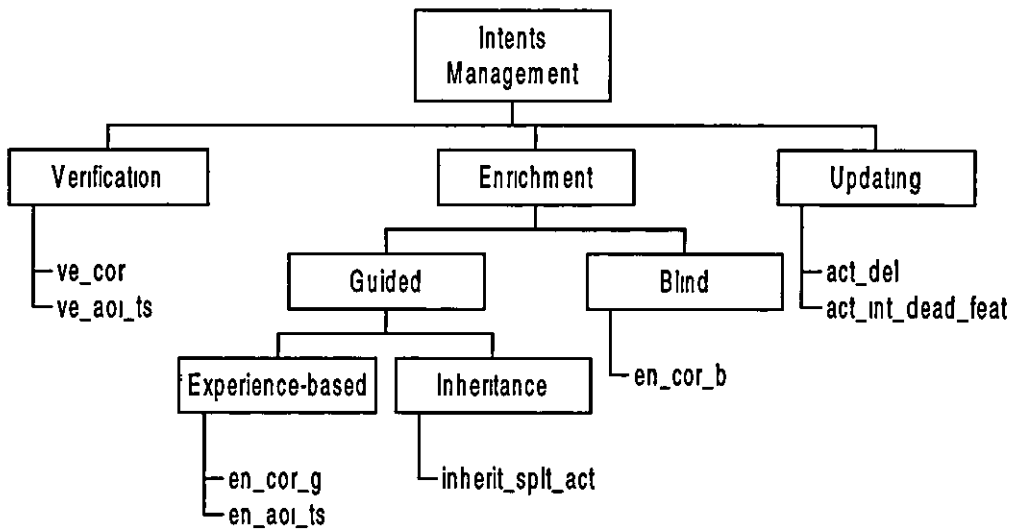


Figure 7-3: A Classification of Intents Management Rules.

Three ways can be identified to manage FbDI validation (Figure 7-3): *verification*, *enrichment* and *updating* statements. The figure also exemplifies some of the rules which are detailed in chapter 8.

7.4.1 VERIFICATION STATEMENTS

The *verification* statement is used to check if the assigned FbDI in the model complies with its conditions. Otherwise, it can lead to its removal from the model.

The general outline of *verification* rules is depicted in Figure 7-4 where:

- “!” means existence or true/active;
- “~” means absence or false/inactive;
- “FbDI” is the target feature-based designer’s intent;
- “Cond” is a condition being tested.

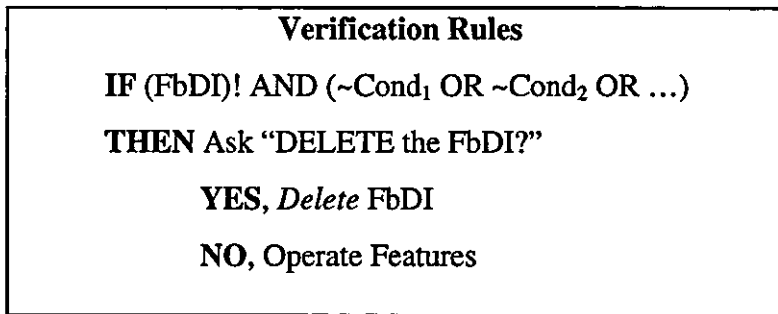


Figure 7-4: An Outline of *Verification* Rules.

7.4.2 ENRICHMENT STATEMENTS

The *enrichment* (or feature intent recognition) statement works in the opposite direction to a *verification* in that it analyses a set of conditions and assigns its findings to the model (automatically or assisted by the designer).

Verification statements are basically invalidity tests that inactivate a FbDI as soon as any of its conditions are violated. *Enrichment* statements do the opposite by considering a set of conditions that suggest a FbDI to be assigned to the model. However, two ways can be identified to perform such a search: via *guided* rules or via *blind* rules of *enrichment*.

Blind rules of *enrichment* involve trying a FbDI relationship against all possible situations using a minimal condition set and leaving the confirmation task to the user. This approach is likely to identify an important FbDI but also leads to a tedious confirmation task.

Guided *enrichment* rules search for FbDI's where they are more likely to occur through rules that include basic conditions plus other conditions identified by experience. Although a less tedious confirmation process follows, it is possible that a FbDI can be omitted from the model due to an inaccurate or missing rule.

Guided *enrichment* can be further classified (Figure 7-3) into *experience-based* guided enrichment when they are isolated rules as mentioned above and

inheritance-based guided enrichment when the rules are embedded and dependent on other (mainly VDI) reasonings.

For instance, it is sensible to think that features that were split from another tend to inherit the former's FbDI's (Figure 8-29).

The general outline of *enrichment* rules are depicted in Figure 7-5:

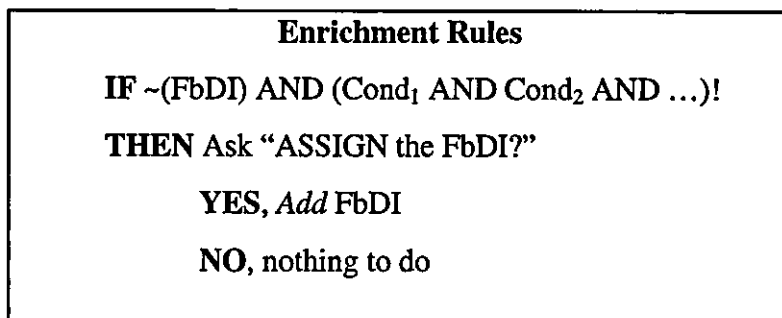


Figure 7-5: An Outline of *Enrichment* Rules.

7.4.3 UPDATE STATEMENTS

Verification and *enrichment* are responsible for *deleting* and *adding* FbDI relationships to the model, respectively.

In addition to *enrichment* and *verification* statements, other rules are necessary to help the management of FbDI's. These are called *updating* rules (Figure 7-3) and they consider the status of the features involved in the relationship and activate or inactivate FbDI's accordingly. Examples of *updating* rules include:

- *Act_del*, if a feature that previously made obsolete or deleted another feature is subsequently inactivated then the latter should be reactivated and the corresponding VDI inactivated.
- *Act_int_dead_feat*, if a FbDI exists for inactive features then it should be inactivated.

7.5 ACTIVE, INACTIVE AND INTENTIONAL STATUS

The process of design can cause the representation to go through many intermediate stages. One approach to help cope with these intermediate stages is to define an *intentional* or dormant status (which is compatible with the intent-driven terminology).

The idea of intentional features have been already introduced (Tomiyama90, Rossignac91): "Intentional features, originally identified by the designer, should not be confused with their geometric embodiments which can vary as the model is edited". This distinction is essential for representing and interrogating invalid features and helps the tracing of feature evolution through the life-cycle of a design model.

Similarly, the validation framework makes use of the *intentional* status and thus features (or more precisely, their *volumetric intentions* - FV + FN) and FbDI's are kept in the framework in one of three possible status:

- The *active* status, which accounts for all those features and FbDI's that represent the actual model. After the reasoning is finished, the active status identifies validated (non-invalid) features and validated FbDI's.
- The *inactive* status, which refers to all features and FbDI's that were deleted by the reasoning of interacting features or by the user and are not affecting the actual model. Inactive features and FbDI's explicitly deleted by the user will not become active in the future and can be effectively removed from the database.

An *inactive* FbDI means that the possibility of existence of this FbDI was considered before, presented to the user and discarded. In this case, an *inactive* FbDI would have been created to flag the discarded attempt and will not be considered subsequently so long as the conditions do not change.

- The *intentional* status accounts for dormant or intermediate situations. *Intentional* features are those that were made obsolete by another feature. Their *volumetric intentions* still affect the model but are encompassed by the *volumetric intention* of another feature. If this second, volumetrically encompassing, feature is removed the former *intentional* feature should be activated.

Similarly, feature interaction cases can be of *active*, *inactive* or *intentional* status.

- An *active* feature interaction is that (possibly recently determined by the scenario identification methodology) which is considered by the *simply volumetrical* and *complex* VDI reasoning sets. These reasonings can change the status of the interaction to *inactive* or *intentional*.
- *Intentional* feature interactions are those interaction cases that should be intentionally left unoperated due to their meaning, unforeseen reasons or other restrictions. These situations usually happen when (a) an arrangement of (subjoint VI or conjoint FI) interacting features occurs that is interesting for an optimisation that an application can perform considering their original interaction or (b) to facilitate and simplify the model.

For instance, if there is no advantage in splitting a *hole* on the periphery of and entering a *pocket*, the *enter* subjoint VI interaction can be defined as *intentional* which later, through another reasoning, can give rise to a “cut-out” case (see Figure 4-8 in section 4.4.3.2.3). Also, a *non-through hole* that *crosses* a *slot*, although it could be split due to redundancy VDI reasons, should be signalled as an *intentional cross* otherwise there will be accessibility problems in drilling the internal *hole*.

- An *inactive* interaction means one of two things: either the interaction was properly processed and a new scenario emerged so that the originating

interaction no longer exists, or the possible use of that interaction case has already been taken into account and should not be tried again.

7.6 INTENT MANAGEMENT PRIORITY

To accommodate FbDI's beyond *morphological functional* FbDI (MFI) analysis, i.e. to include geometric relational FbDI (GDI) and application-oriented relational FbDI (AOI) reasonings, the priority scheme presented in Figure 7-6 was conceived by extending the MFI reasoning set organisation (Figure 7-2).

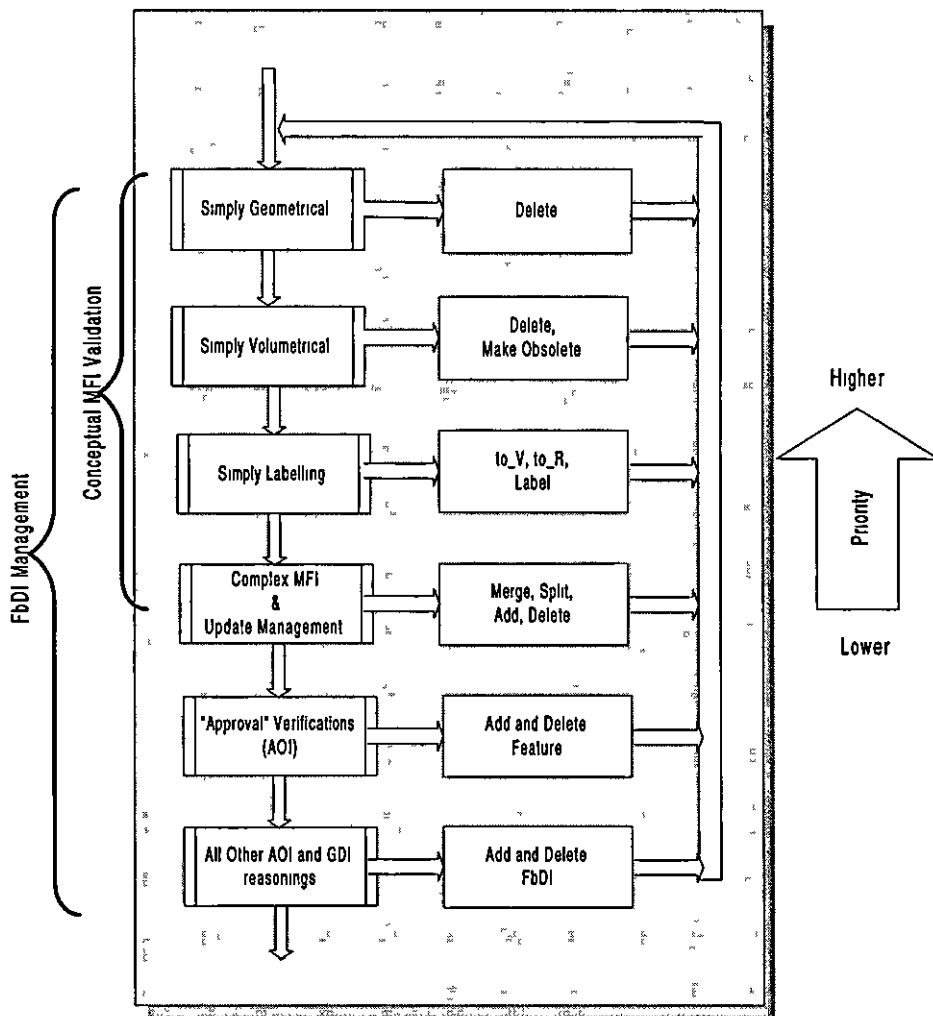


Figure 7-6: The Intent-Driven Conceptual Validation Reasoning.

All the RDI *verifications* and *enrichments* are placed with lower priority than the MFI reasonings but in two levels: a higher level where a newly added feature can be rejected by some reasoning, called here the *approval* phase, such as the *proximity* AOI test (see section 4.4.3.2) usually via a *verification* statement, and lower level which comprises all other AOI and GDI *verification* and *enrichment* rules, simply called the *enrichment* phase. These analysis phases can be turned on and off in the prototype implementation if required.

7.7 UNDERSTANDING THE REASONING ORGANISATION

To help make sense of the priority organisation, a metaphoric parallel can be established between these reasonings and a linguistic analysis. Linguistic analysis is achieved in four phases: lexical, syntactical, semantical and contextual analysis.

It could be said that *simply geometrical* and *simply volumetrical* reasonings correspond to a lexical feature-based analysis where the correct use of letters (feature elements such as its *volume*, its faces and its GSM evaluation) to form words (feature definitions) are analysed.

Simply labelling corresponds to a syntactical feature-based analysis where the correct disposition of words (the feature definition, in particular its *label*) of the vocabulary (feature library) on a phrase (feature model representation) are analysed. It assumes a lexically correct (valid) set of letters (feature elements).

Complex MFI reasonings correspond to a semantic feature-based analysis where correct association (feature interactions) of words (features) to produce a meaningful (conceptually validated representation) phrase is analysed. It assumes a syntactically correct (valid) phrase (feature model representation).

Update management ensures that the phrase (feature model representation) is simplified such as when a “double negative” is modified in a linguistic analysis.

Table 3: A Metaphoric Parallel (Feature-based Model and a Language).

<i>Feature-based Models</i>	<i>Language</i>
Feature elements (e.g. edges, vertices, faces, <i>volume</i> and evaluated GSM)	Letters (e.g. a, b, c, α , β , χ)
Feature's <i>label</i> (e.g. <i>slot</i> , <i>pocket</i> , and <i>step</i>)	Words (e.g. cat, dog, and horse)
Feature-based model representation	Phrase
<i>Simply geometrical</i> and <i>simply volumetrical</i> reasoning	Lexical analysis
<i>Simply labelling</i> reasoning	Syntactical analysis
<i>Complex MFI</i> reasoning	Semantical analysis
<i>Update</i> management	Simplifications
<i>Approval</i> management (e.g. thin-wall tests)	Has meaning, but not for the target context
Other AOI and GDI reasoning	Elaborate context analysis

The *approval* reasoning analyses the model for cases where the last manipulation can be immediately rejected such as when an extra word is syntactically correct but adds no meaning to the context of the phrase.

All further reasoning performs a more elaborate contextual analysis. Contextual feature-based analysis occurs when the meaning (conceptual validity) of the phrase (feature model representation) to the context (application) being considered is analysed. This is done by considering a much broader spectrum of FbDI's, beyond MFI's, that in turn also consider application-dependent criteria. It assumes a semantically correct (conceptually MFI valid representation) phrase (feature model representation).

7.8 SUMMARY

This chapter has introduced the *verification* and *enrichment* statements, which are responsible for FbDI management and the *active*, *inactive*, and *intentional* status of features, FbDI's and interactions. These characteristics help implement the validation concept.

The way in which elements of a feature-based vocabulary are bound together to perform conceptual representation validation has been described. The invalidity tests are organised in a hierarchical priority and are used to perform validation.

It has also been shown that the priority is reasonable if a metaphoric comparison to a linguistic analysis is accepted.

The next chapter describes how these ideas have been implemented in a prototype system.

8. IMPLEMENTATION

The application of the elicitation process produced a classification and a taxonomy for feature interactions, feature-based operations and FbDI's. To validate this development methodology and philosophy, a working prototype was implemented that is able to identify intents and feature interactions, apply modelling operations, perform verifications and enrichments and deploy revalidation operations (some of these automatically).

8.1 A PROTOTYPE IMPLEMENTATION

A prototype system known as **FRIEND**, an acronym for Feature-based validation Reasoning for Intent-driven **EN**gineering Design, has been implemented. The acronym was chosen to emphasise the following aspects:

- It is a system centred on the Feature concept, their elements and properties.
- It is a framework for conceptual validation analysis that was implemented using an expert system shell for its validation Reasoning.
- It is Intent-driven which should produce a more forgiving environment.
- It is intended to accommodate various **EN**gineering-related Design disciplines (applications).

8.2 RESOURCES

FRIEND has been implemented on a PC-compatible Pentium™ 100Mhz with 24 MB of EDO RAM running Microsoft® Windows 95™ operating system. The applications used to develop the prototype system are wxCLIPS and Microstation® 95's MODELER.

8.2.1 WxCLIPS

WxCLIPS 1.61, 32 bit version, is a graphical development environment (Figure 8-1) that extends CLIPS' functionality with hundreds of functions to allow a graphical user interface to be constructed. It was developed by Julian Smart at the AIAI (Artificial Intelligence Applications Institute), which is part of Edinburgh University.

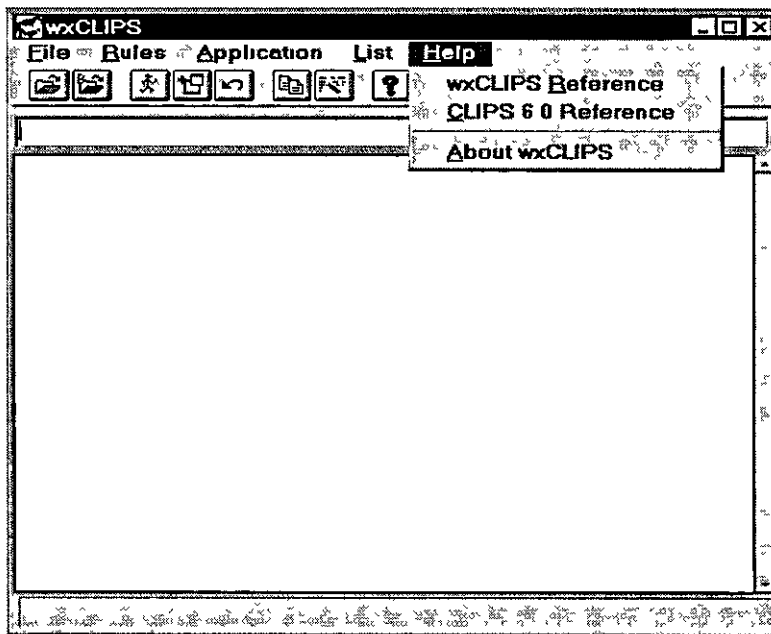


Figure 8-1: wxCLIPS Environment Screen.

CLIPS was designed at NASA/Johnson Space Center and is an acronym for “C Language Integrated Production System” which recalls its origins. CLIPS 6.0 is a forward chaining expert system shell and a multiparadigm programming

language that provides support for rule-based, object-oriented and procedural programming. It is a parenthetical language and most functions and commands use a prefix syntax (Figure 8-2, Giarratano93, Donnell94). Figure 8-2 provides a parallel between CLIPS syntax and a pseudo-code in an attempt to show the meaning of some of CLIPS code.

PSEUDO-CODE	CLIPS Syntax
RULE-NAME "example" IF (Comment: Oven Operation Monitoring) "OvenPower" is "ON" AND "Pressure" > 5 AND ("Simulation" fact does NOT exist OR "Operator_Status" is "trainee") THEN Create "Urgency" fact Calculate "risk_factor" Turn OFF OvenPower END-RULE	<pre>(defrule "example" : Oven Operation Monitoring ?o_p <- (OvenPower ON) (test (> Pressure 5)) (or (not (Simulation)) (Operator_Status trainee)) => (assign (Urgency)) (risk_factor) (retract ?o_p) (assign (OvenPower OFF)))</pre>

Figure 8-2: CLIPS Syntax.

CLIPS 6.0 provides COOL, which is "CLIPS Object Oriented Language", and a comparison between COOL syntax and a pseudo-code for a class definition is presented in Figure 8-3.

PSEUDO-CODE	COOL Syntax
CLASS-NAME "Intent" Super-Class "Relation" Integer "n" (Valid Interval 1-3, Default Value 2) Array of 6 words "a_6_w" END-CLASS	<pre>(defclass Intent (is-a Relation) (slot n (type NUMBER) (allowed-symbols 1 2 3) (default 2)) (multislot a_6_w (type SYMBOL) (cardinality 6 6)))</pre>

Figure 8-3: COOL Syntax.

Simplified versions of class and rule definitions will be presented using CLIPS and COOL syntax. More specific details that are exclusive to the CLIPS and COOL implementations will be omitted. The understanding of Figure 8-2 and Figure 8-3 should be sufficient to understand the listings and the accompanying explanations in the following sections.

8.2.2 MICROSTATION

Microstation® 95 is a CAD system produced by Bentley Systems that has an extra module based on the ACIS® 1.7 geometric solid modeller and is called **MODELER 1.0**.

8.3 MODULES

For implementation reasons **FRIEND** (which follows a DbF approach, Figure 2-1) was divided into two main modules that reflect the two items of software used (Figure 8-4): **FRIEND-KBS** (implemented using wxCLIPS) and **FRIEND-VIEW** (implemented using **MODELER**).

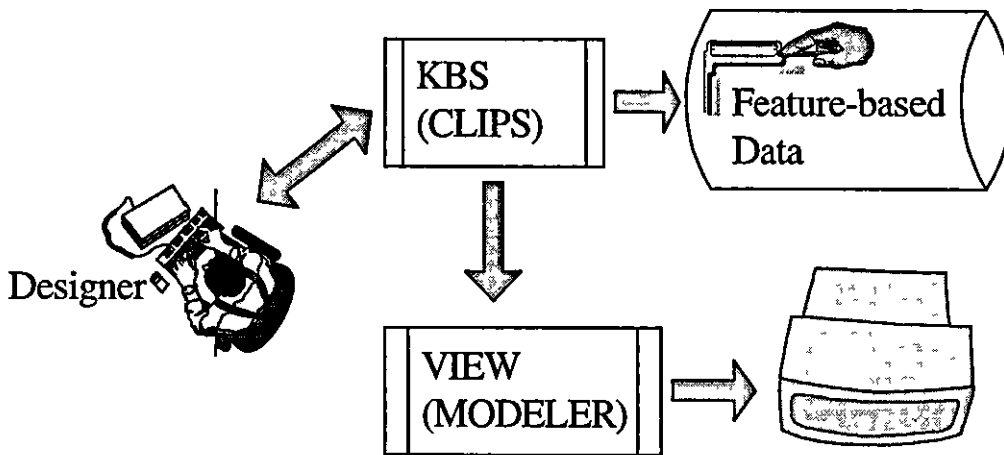


Figure 8-4: FRIEND Modules.

8.3.1 FRIEND-VIEW

FRIEND-VIEW is the module responsible for the geometric evaluation and visualisation of the feature-based model. It converts features into their producing volumes (solid primitives), position and orientates the volumes and

performs chamfering and Boolean operations accordingly. The model can be visualised either in a Boolean operated or in a Boolean unoperated form (see Figure 9-5). The unoperated form shows all feature *volumes* without actually performing the Boolean operations and was made available because the operated invalid model frequently looks like the operated valid one, confusing the viewer.

FRIEND-VIEW was prototyped using Microstation's Basic language, which, although slow and limited, is simple to use and code, easy to understand and suffices for the prototype implementation. FRIEND-VIEW is initiated by pressing the **FRIEND** icon at the top left-hand corner of MODELER's environment (see the "dog" icon in Figure 8-5) which starts the communication protocol between the modules FRIEND-KBS and FRIEND-VIEW.

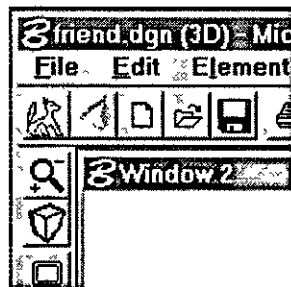


Figure 8-5: FRIEND-VIEW Icon.

8.3.2 FRIEND-KBS

FRIEND-KBS is the module responsible for the reasonings. Figure 8-6 presents the architecture of the FRIEND-KBS module. It details the framework presented in Figure 7-1 and shows that the data information used for the reasoning are basically of two types: the feature library (including the template of each feature) and the reasoning groups presented in section 7.6 divided according to the type of FbDI involved. The figure also shows that three types of information are generated and maintained by the reasoning: the feature, the feature interactions and the FbDI's of the model. These three items of

information come in three different flavours: intentional, valid/active or invalid/inactive.

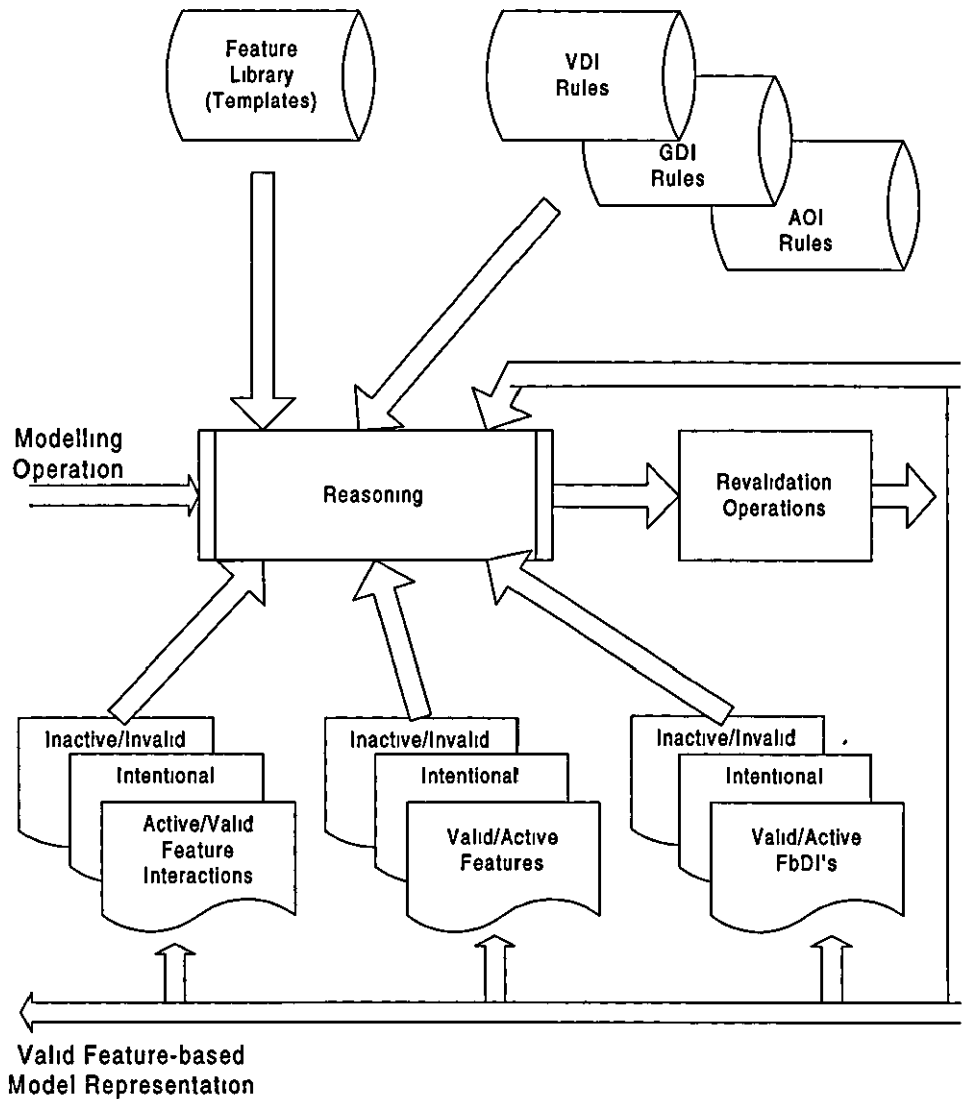


Figure 8-6: FRIEND-KBS Architecture.

The FRIEND-KBS interface has three main areas (Figure 8-7):

- A text area, in the lower part of the window, where additional information is displayed as the reasoning is being carried out;
- A button area, for activating fast actions such as “Show Validated Part” which forces the update of the model’s visualisation in FRIEND-VIEW;

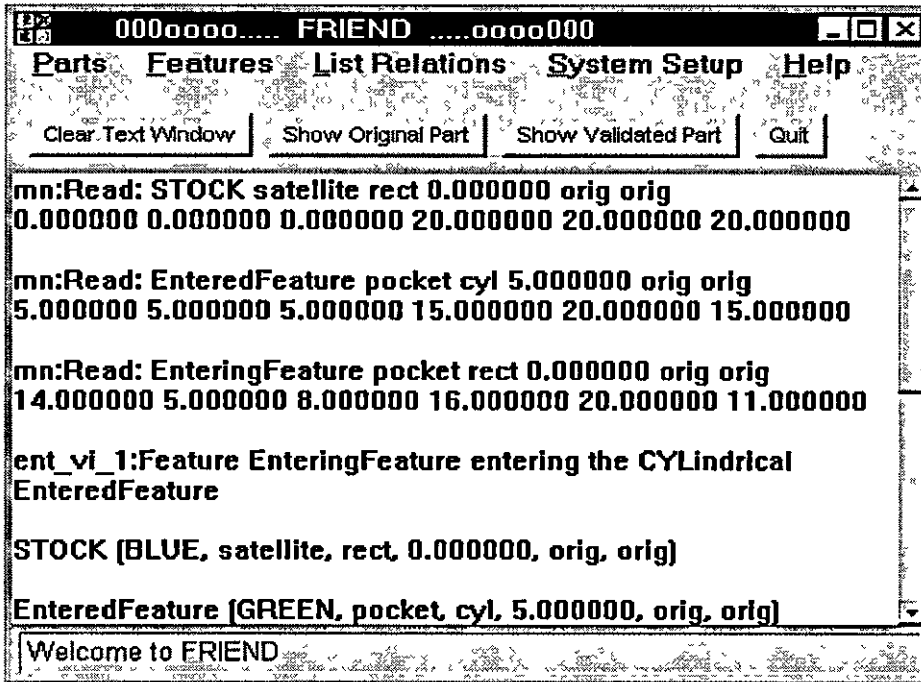


Figure 8-7: FRIEND-KBS Module.

- A menu area, where various options to manipulate the model are available:
 - To manipulate (Figure 8-8), such as add feature (Figure 8-9);
 - To manipulate a part file (see section 9.2) such as load and save part file (Figure 8-10);
 - To list relations including interaction cases and FbDI's (Figure 8-11).

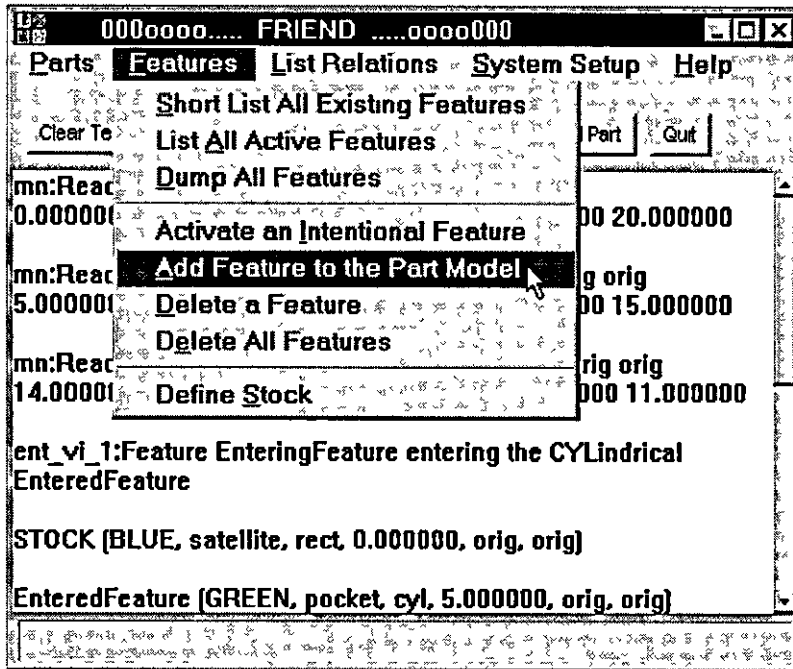


Figure 8-8: Feature Manipulations in FRIEND.

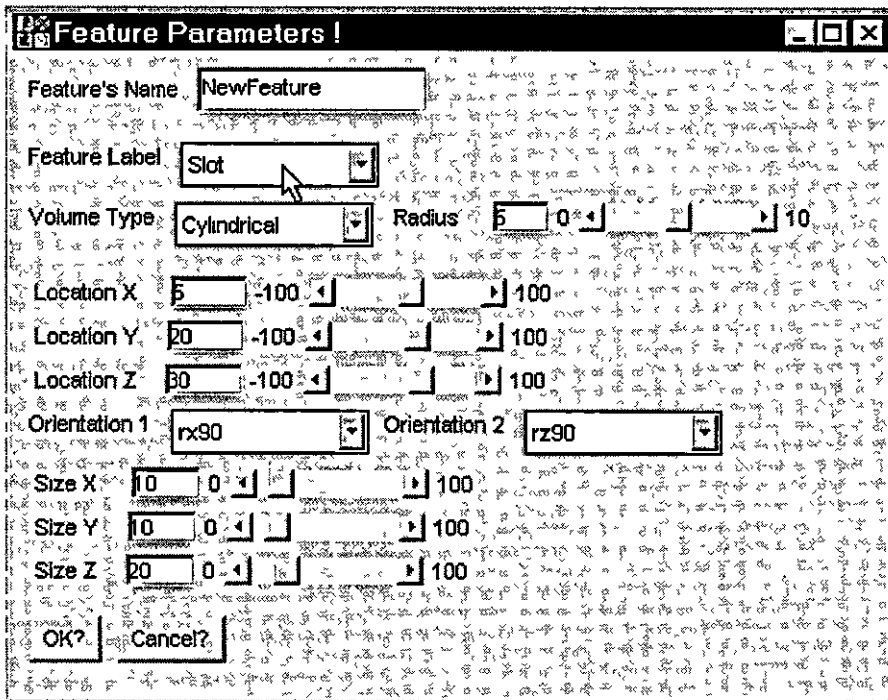


Figure 8-9: Add Feature Option.

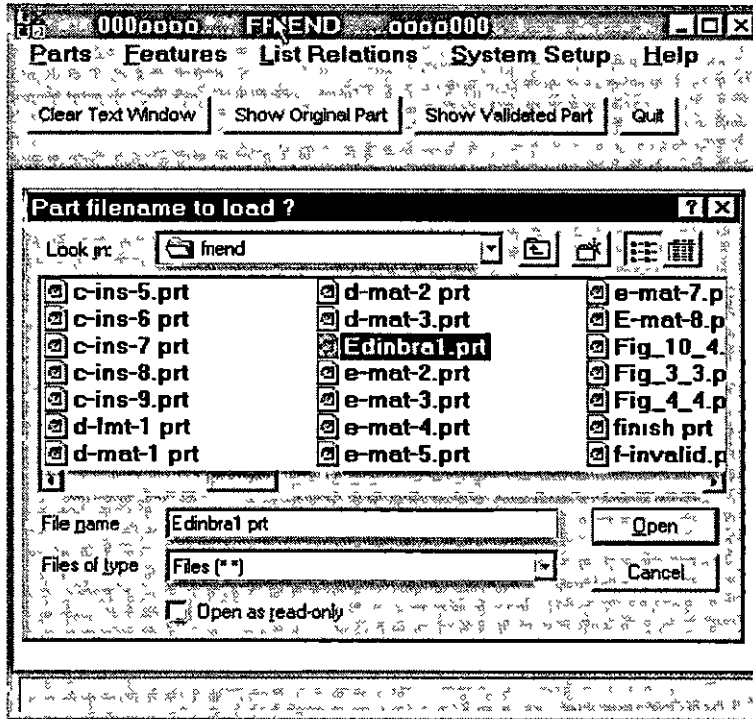


Figure 8-10: Load a Part File Option.

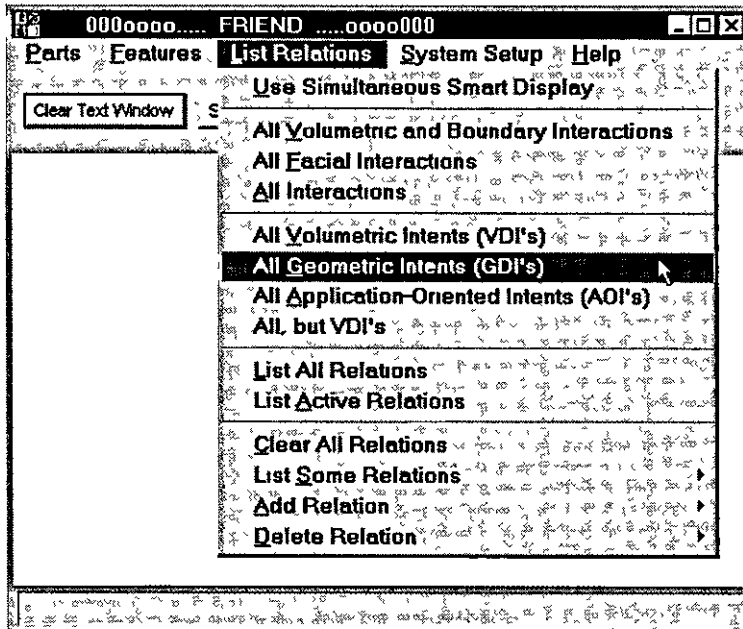


Figure 8-11: FRIEND-List Relations Option.

8.4 DATA STRUCTURES

8.4.1 REPRESENTING INTENTS AND INTERACTIONS

It has been suggested that a feature-based model should be more than a list of active features and should also include all relations such as constraints (Kraker97) and interactions (Kim96). In addition, instead of evaluating a feature interaction query every time it is requested, a feature interaction scenario can be produced between an incoming feature and the existing active features in the model. The result can be stored as active relations that can be used by the reasoning when needed.

```
(defclass RELATION
  (slot status
    (type SYMBOL)
    (allowed-symbols active inactive intentional)
    (default active))
  (slot go_further ; Boolean
    (type SYMBOL)
    (allowed-symbols YES NO)
    (default YES))
  (slot has_master
    (type SYMBOL)
    (allowed-symbols YES NO)
    (default YES))
  (slot master
    (type INSTANCE-NAME)
    (default ?NONE))
  (slot n_slaves
    (type INTEGER)
    (default 1))
  (multislot slaves
    (type INSTANCE-NAME)
    (default ?NONE))
  ; See subclasses for code and type constraints
  (slot type
    (type SYMBOL))
  (slot code
    (type SYMBOL))
)
```

Figure 8-12: RELATION Class.

The relations of interest in this research are interactions and FbDI's. All relations share the same basic principle of establishing a directional and meaningful link between elements. Therefore, a class called RELATION (Figure 8-12) and its two sub-classes, called INTENT (Figure 8-13) and INTERACTION (Figure 8-14) were defined.

The RELATION class establishes a directional link by defining a *master* of the relation and a list of *slaves*. Elements referred to are features or (index to) a feature's (bounding box) face. The specific meaning of the relation is defined by the type and code, which according to the sub-class, have different lists of allowed possibilities.

For instance, an INTERACTION can be typed VI and coded conjoint while an INTENT can be typed VDI and coded split_into.

The implementation does not accommodate relations that carry a reference or parameter but it is acknowledged that such a resource should be considered in future implementations. In addition, the labelling VDI is kept as the feature's *label* in the FEATURE class and not as a (unary) INTENT object.

```

(defclass INTENT
  (is-a RELATION)

  (slot code
    (allowed-symbols
; MORPHOLOGICAL DI's (MFI's)
; VDI - Volumetrical Morphological DI's
split_into merged_from
deleted_by
obsoleted_by

; GEOMETRICAL DI's (GDI's)
; HieGDI - Hierarchical Geometrical Relational DI's
nested@bot nested@side

; PosGDI - Positonal Geometrical Relational DI's
concentric
coplanar

; OriGDI - Orientational Geometrical Relational DI's
parallel perpendicular angular
against co-linear
coEAD

; StrGDI - Structural GDI's
pattern
ax_symmetry rd_symmetry
co-radius

; APPLICATION-ORIENTED INTENTS (AOI's)
t_slot
c_bore c_sink
x_feat e_feat
cut_out
precede succeed ))

  (slot type
    (allowed-symbols VDI HieGDI PosGDI StrGDI OriGDI AOI)
    (default VDI))
)

```

Figure 8-13: INTENT Class.

In the case of the sub-class INTERACTION, an interaction VI could be linked to many other interactions (BI or FI) and therefore pointers to other interactions are included in the class (Figure 8-14).

```

(defclass INTERACTION
  (is-a RELATION)

  (slot code
    (allowed-symbols
     ; VI Volumetric Interactions
     connected disconnected
     conjoint adjoint disjoint overlap
     near far
     cross enter crossed entered general

     ; BI Boundaring Interactions
     limit limited contain contained

     ; FI Facial Interactions
     match inside outside

     unknown )
    (default unknown))

  (slot type
    (allowed-symbols VI FI BI)
    (default VI))

  (slot n_elements ; 8-08-96
    (type NUMBER)
    (allowed-numbers 0 1 2 3 4 5 6)
    (default 0))

  (multislot elements
    (type INSTANCE-NAME) ; pointer to another INTERACTION
    (default ?DERIVE))

  (slot GSM_confirms
    (type SYMBOL)
    (allowed-symbols YES NO DONO)
    (default DONO))

  )

```

Figure 8-14: INTERACTION Class.

Because the implementation of the interaction identification was done using the bounding box envelope, it was necessary to confirm some geometric interactions at the level of the actual feature volume (FAV), boundary (FAB) or face (FAS). To identify if this confirmation had been carried out, a *GSM_confirms* flag was included.

8.4.2 REPRESENTING FEATURES

A feature is implemented by defining a FEATURE class (Figure 8-15) which stores a *sequence* number; a feature name as its *id*; the feature *status*; the feature *label*; feature *nature*, which can be *add* for additive or *rem* for subtractive volumes; feature *volume* which can be *cyl* for cylindrical or filleted features and *rect* for quadrangular ones; feature *radius* that stores the radius of the cylinder or the chamfer; feature *main axis*; two rotations used to specify the feature's orientation, each rotation is defined by an Euclidean axis and an angle of rotation.

In addition, every feature has an associated bounding box *envelope* (Figure 8-15). The *envelope* is stored as an object of the BBOX class (Figure 8-16) defined by the minimum and maximum vertex coordinate values. In the prototype implementation, all the geometric analyses use the *envelope* instead of the actual evaluated volume of the feature.

Therefore, the actual volume is projected onto the envelope faces that acquire characteristics such as face *code*, *token* and *profile* (Figure 8-17). Figure 8-17 presents a *step* feature, the feature *volume* with the identification of its *codes* and *tokens* followed by the identification of every face's projection onto a rectangular bounding box envelope.


```

(defclass FEATURE

(slot sequence (type INTEGER)
  (default 1))
(slot id (type SYMBOL))
(slot status (type SYMBOL)
  (allowed-symbols active inactive intentional)
  (default inactive))
(slot label (type SYMBOL)
  (allowed-symbols pocket hole
    slot slot_tru step notch
    slab gap hollow satellite)
  (default pocket))

;----- Volumetric Intention (FV + FN + orientation) -----
(slot nature (type SYMBOL)
  (allowed-symbols add rem)
  (default rem))
(slot volume (type SYMBOL)
  (allowed-symbols cyl rect)
  (default rect))
(slot radius (type NUMBER)
  (default 0.0))
(slot axis (type INSTANCE-NAME)
  (default-dynamic (make-instance (gensym*) of POINT)))
(slot rotation_1 (type SYMBOL)
  (allowed-symbols orig rx90 rx180 rx270
    ry90 ry180 ry270 rz90 rz180 rz270)
  (default orig))
(slot rotation_2 (type SYMBOL)
  (allowed-symbols orig rx90 rx180 rx270
    ry90 ry180 ry270 rz90 rz180 rz270)
  (default orig))

; ----- The Envelope (Bbox + Code + Profile + Token) -----
(slot envelope (type INSTANCE-NAME) ; of BBBOX
  (default-dynamic (make-instance (gensym*) of BBBOX)))
(slot face_nature (type INSTANCE-NAME) ; of FC_NATURE
  (default-dynamic (make-instance (gensym*) of FC_NATURE)))
)

```

Figure 8-15: FEATURE Class.

```

(defclass BBOX
  (slot owner
    (type INSTANCE-NAME) ; Pointer to a feature
    (default ?DERIVE))
  (slot Pmin
    (default-dynamic (make-instance (gensym*) of POINT)))
  (slot Pmax
    (default-dynamic (make-instance (gensym*) of POINT)))
  )

```

Figure 8-16: BBOX Class.

For the prototype implementation, the face's *code* suffices to identify the feature *label*.

A face's *code* is the identification of the expected face's contribution to the boundary model of the part and it can be :

- *Virtual (V)*, if the face is expected not to produce a face on the solid model or if there is no material on either side of the face and therefore it is access for a tool.
- *Real (R)*, if the face is expected to produce or imprint a face on the evaluated solid model or if there is material on one side of the face and not the other and therefore it is a face to be machined.

In addition to the face *codes*, *tokens* and *profiles* are assigned to every projection of the feature volume onto the envelope's faces.

A face's *token* identifies the face's function. It can be said that a *token* is a specialised version of the face's *code*. However, it is difficult to reason where a *real* face is a *bottom* or a *side* face without having the understanding of the designer. Face *tokens* can be:

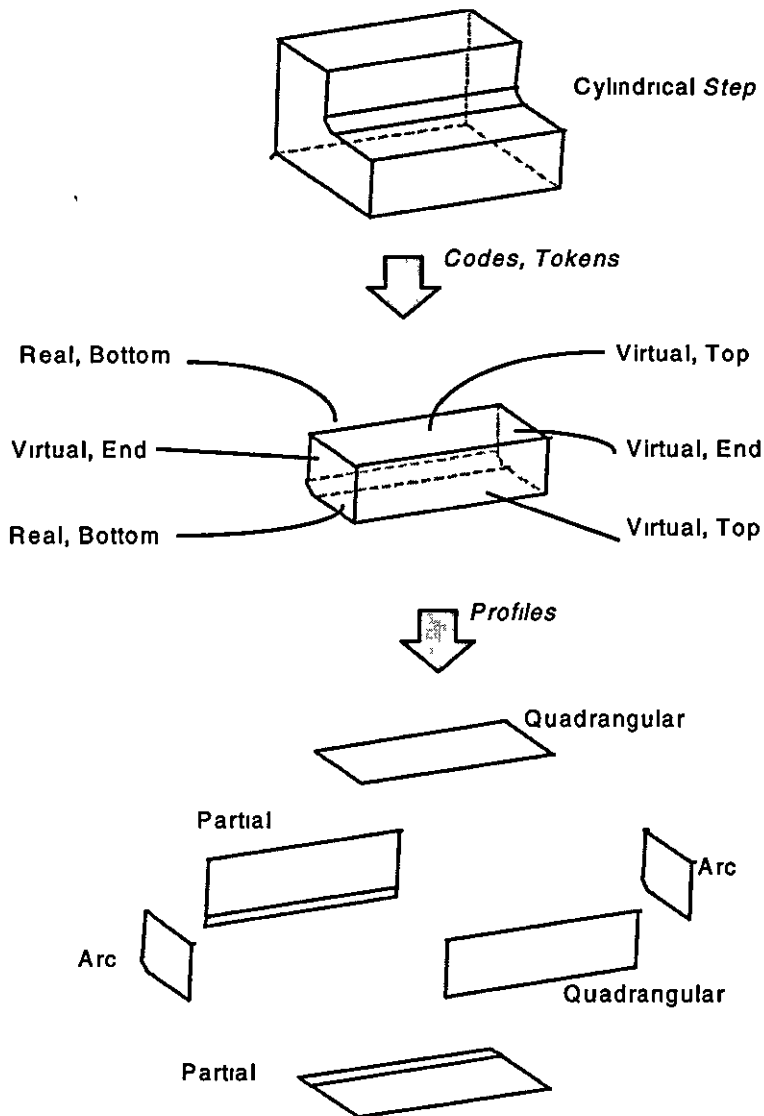


Figure 8-17: Feature Face Characteristics.

- *Top (T)*, a *virtual* face that identifies a tool's External Access Direction (EAD, Gindy89), opposite to a *bottom* face.
- *End (E)*, a *virtual* face opposite to another *end* face.
- *Bottom (B)*, a *real* or *virtual* face opposite to a *top* face.
- *Side (S)*, a *real* face opposite to another *side* face.

```

(defclass FC_NATURE
  (multislot code
    (type SYMBOL)
    (allowed-symbols R V)
    (cardinality 6 6)
    (default ?DERIVE))
  (multislot profile
    (type SYMBOL)
    (allowed-symbols Q C L A)
    (cardinality 6 6)
    (default ?DERIVE))
  (multislot token
    (type SYMBOL)
    (allowed-symbols S B T E )
    (cardinality 6 6)
    (default ?DERIVE))
  )

```

Figure 8-18: FC_NATURE Class.

A face's *profile* identifies the shape that the feature volume (FV) projects onto the envelope's face. *Profile* projections can be:

- *Arc (A)*, when a surface containing an arc is produced.
- *Quadrangular (Q)*, when a quadrangular face is produced that coincides with the envelope's face.
- *Line (L)*, when a line is produced by the touching of the projecting FV's onto the envelope's face and is lateral to a face with a *C profile*.
- *Partial (P)*, when a quadrangular face occurs, but it only partially covers the envelope's face and is lateral to a face with an *A profile*.
- *Circular (C)*, when a circular face projection occurs.

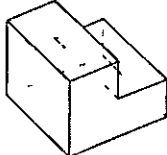
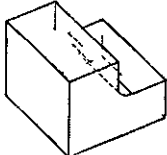
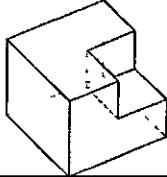
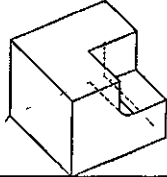
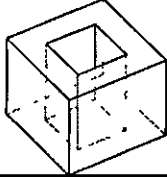
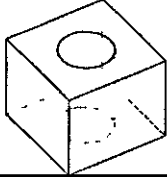
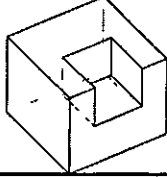
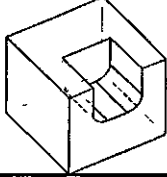
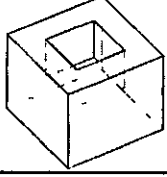
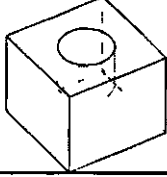
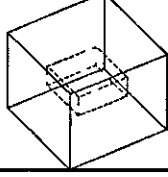
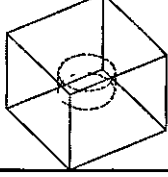
Feature Label	Quadrangular Volume		Cylindrical Volume	
	Standard Position	Bbox Faces 1 2 3 4 5 6	Standard Position	Bbox Faces 1 2 3 4 5 6
Step (EAD's = 4)		<u>VVVRRV</u> QQQQQQ ETTBBE		<u>VVVRRV</u> AQQPPA ETTBBE
Notch (EAD's = 3)		<u>VVVRRR</u> QQQQQQ TTTBBB		<u>VVVRRR</u> AQQPPA TTTBBB
Hole (EAD's = 2)		<u>RRVVRR</u> QQQQQQ SSTBSS		<u>RRVVRR</u> LLCCLL SSTBSS
Slot (EAD's = 2)		<u>VRVRRR</u> QQQQQQ TSTBSB		<u>VRVRRR</u> APQPPA TSTBSB
Pocket (EAD's = 1)		<u>RRVRRR</u> QQQQQQ SSTBSS		<u>RRVRRR</u> LLCCLL SSTBSS
Hollow (EAD's = 0)		<u>RRRRRR</u> QQQQQQ SSTBSS		<u>RRRRRR</u> LLCCLL SSTBSS

Figure 8-19: Features and their Codes, Profiles and Tokens.

Figure 8-19 shows some example features in their standard orientation beside a list of characteristics for every face of the envelope. The first line of the "Bbox faces" column identifies the face codes; the second, the face profiles and; the last the face tokens.

In Figure 8-19, a template pattern of *virtual* and *real* codes (presented in bold underlined italic letters) is used to identify the feature *label* of that type. The features are presented in their standard position with no changes to the default orientation.

Tokens can also be used to perform *labelling* reasoning (cf. Silva90) but, for the domain chosen, *codes* seem to suffice and *tokens* are assigned to the found label/orientation as are *profiles*. *Tokens* are used to ease some reasonings but, in this implementation, are dependent on the *code* and orientation determination.

What has previously been called a feature's *End* face in chapter 5 (Figure 5-10) refers to an envelope's face with a *virtual* code (those that can have a *Top* or *End* token).

8.5 INTENTS MANAGEMENT IMPLEMENTATION

Some FbDI's have one *enrichment* rule and its dual *verification* rule while others perform their analysis through many *verification* statements that represent many different situations.

The rules proved to be a little more complicated than expected and thus slightly different from the outlines given in Figure 7-4 and Figure 7-5, although the principle still applies. There are two reasons for this:

- sometimes it is necessary to consider an intent or interaction where the feature can be considered either a master or slave of that relation.
- sometimes many situations have the same actions, and are implemented in one rule only.

There are FbDI's that do not apply to some specific features and some FbDI's have specialised versions when facing specific (pairs of) features (for instance

coaxiality can be defined as a *concentricity positional* GDI between *hole* features). Therefore, once more, a FBM system should consider those FbDI's that are more suitable for the target domain and avoid overlapping FbDI's.

8.6 FEATURE INTERACTION IDENTIFICATION IMPLEMENTATION

The methodology of feature interaction identification has been implemented and bound to FbDI's using feature envelopes (bounding boxes) in order to achieve reasonable efficiency. It was found that *feature interaction* cases can be accurately and quickly predicted through their envelope and references to the actual volumes and faces. These references allow the system to effectively apply such determination schemes at a lower level if required.

High levels of identified interaction act as filters or approximations for lower levels of interaction cases if they are not used promptly for some specific reasoning before lower levels of interaction are identified. Thus, the implementation is facilitated and accelerated because of this filtering aspect.

The binding of each interaction case with an action via rules is the process that allows **FRIEND** to perform its task – feature-based representation validation.

8.7 FEATURE OPERATIONS IMPLEMENTATION

The following are the operations implemented in **FRIEND**:

- Analysis operations;
- Add FbDI, delete FbDI, add feature and delete feature modelling manipulation operations;
- Add FbDI, delete FbDI, add FV, delete FV, make obsolete, make active, split, merge and label revalidation manipulation operations.

A simple way to delete a feature is to make it *inactive* as well as making all feature interaction cases that refer to it *inactive* or *intentional* (which is done automatically via the intent management reasoning).

It should be mentioned again that the most recently requested *operation* (type and operation itself, e.g. *editing-add intent*) is not considered by the validation reasoning (which works the same way for all operations) although it is recognised that its use could ease the reasoning. For instance, optimised validations could be triggered according to the *add* or *delete* operation. However, this was not necessary in the framework established here.

8.8 PRIORITY IMPLEMENTATION

Figure 7-2 shows that to implement the loop of reasoning in **FRIEND**, the validation reasoning consists of the various reasoning sets organised in a hierarchical fashion (as implied by the right-hand side of the figure). There is also a priority relationship among the situations related to the figure and the feature interaction identification level (Volumetrical, Boundary and Face). The reasoning goes deep into the interaction level if it can not reason with the information and interaction already available, and this is another reason why the framework in Figure 7-2 is a loop.

The implementation of the reasoning sets and the priority arrangement is achieved using the "salience" facility of CLIPS. A rule's salience identifies its priority. A rule with higher salience is selected to be executed (fired) if compared to another active rule. Groups of rules were assigned different salience values and placed in various files (to emphasise their meaning and reasoning set).

In essence, the priority scheme suggests that, after identifying the feature interaction case (at an appropriate level - initially volumetrical), some basic

volumetric reasonings analyse the model searching for obvious mistakes of placement and *nature* (regardless of the feature's *label*).

After correcting basic mistakes, a set of reasonings would guarantee that all features are correctly labelled (helped by further feature-interaction analysis) because subsequent reasonings will perform more complex analyses (possibly application-dependent ones) that will subsume correct labelling and none basic volumetric misrepresentation.

8.9 REASONING SETS EXAMPLES

The implementation has separate rule-based files that reflect the division and hierarchy for the feature-based reasoning sets. *Simply geometrical* reasonings that identify feature interaction cases could have been implemented as rules as well but were implemented as functions for efficiency reasons. Also, the search for a feature's *label*, presented in section 6.3.2 as the *search_label* revalidation operation, that identifies the feature's correct *label* according to its face properties which could be in any orientation compared to the original template was also implemented as a function for efficiency reasons.

Some of the rules are presented below using a simplified version of the CLIPS code where some details of the functions being called will be omitted for clarity reasons.

8.9.1 SIMPLY GEOMETRICAL REASONING

Disjoint BI interactions mean that one feature is contained within another and analysing their FAB will lead to a *near* or *far* case. If *near*, then it is possible that an "internal thin-wall" problem may occur and if *far* and if the feature has no other interaction, it can be interpreted as a *hollow* in the part.

For *adjoint* VI cases there will be a possible merging operation (if a *matching* conjoint FI case occur) or a change on the feature's properties from "blind" to "through" (if an *inside* conjoint FI case occurs, see Figure 5-7).

8.9.2 SIMPLY VOLUMETRICAL REASONING

A *simply volumetrical* reasoning is exemplified below. It is considered a *simply volumetrical* reasoning because it uses the *match* volumetrical interaction (thus, it is +V) but not the *label* of the features (thus it is -L):

```
(defrule mat_vi_1
  (declare (sallience 415))
  (phase direct_related)
  (object (is-a INTERACTION)
    (code match)
    (type VI)
    (master ?mstr)
    (n_slaves 1)
    (slaves ?slv))
  (object (is-a FEATURE)
    (name ?mstr)
    (nature ?n1)
    (status active|intentional))
  (object (is-a FEATURE)
    (name ?slv)
    (nature ?n2&~?n1) ; different natures
    (status active))
  =>
  (if (eq YES (Ask "Are you trying to delete slave feature with master feature ?" ))
    then
      (send ?slv put-status inactive)
      ; Add Intent
      (bind ?new (make-instance (gensym*) of INTENT
        (master ?slv) (slaves ?mstr))) )
    else
      (Tell "I will reconsider last add feature !" )
      (send ?mstr put-status inactive)
      ; Add intent
      (bind ?new (make-instance (gensym*) of INTENT
        (master ?mstr) (slaves ?slv))))
  (send ?new put-code deleted_by)
  (+ [REL] ?new)
)
```

Figure 8-20: A *Simply Volumetrical* Rule Example

8.9.3 SIMPLY LABELLING REASONING

A *simply labelling* reasoning is exemplified below. If a face of a given feature “abuts” and is completely inserted into another feature’s *real* face, then the former must be a *virtual* face. Using reasonings such as this the *labelling* VDI can be maintained.

A function *check_label* compares the template and the realisation of the feature and if it does not match then the *label* is invalid, and the *search_label* revalidation operation (inside *check_label*) will then search for the right match. The *search_label* process is responsible for keeping the label-to-shape relationship matching as defined by the template of every feature’s type. It considers a feature interaction at the face level, not at the volumetrical level (thus it is -V), and immediately affects the feature’s *label* (thus it is +L).

```

(defrule ins_fi_11b
  (declare (salience 305))
  (phase direct_related)
  (object (is-a INTERACTION)
    (code inside)
    (master ?mstr)
    (name ?int)
    (type FI)
    (slaves ?slv)
    (status active))
  (object (is-a INTERACTION)
    (master ?mstr_feat)
    (type VI)
    (n_elements ?n&:(> ?n 0))
    (elements $?fi_list1 ?int $?fi_list2)
    (slaves ?slv_feat)
    (status active))
  (object (is-a FEATURE)
    (name ?mstr_feat)
    (status active)
    (object (is-a FEATURE)
      (name ?slv_feat)
      (status active))
  (test (eq (nth$ ?mstr (send ?mstr_feat get-code)) R))
  (test (or (eq (nth$ ?slv (send ?slv_feat get-code)) R) ; R - R, or
            (eq (nth$ ?slv (send ?slv_feat get-code)) V))) ; R - V
  (test (eq (nth$ ?slv (send ?slv_feat get-profile)) Q) ) ; P(slv) = Q
=>
  (if (or (eq (nth$ ?mstr (send ?mstr_feat get-profile)) A)
          (eq (nth$ ?mstr (send ?mstr_feat get-profile)) Q)
          (eq (nth$ ?mstr (send ?mstr_feat get-profile)) C))
    then (send ?mstr_feat put-code (replace$ (send ?mstr_feat get-code)
      (eval (format t "%d" ?mstr)) (eval (format t "%d" ?mstr)) (create$ V) )))

  (send ?mstr_feat check_label)
)

```

Figure 8-21: A *Simply Labelling Rule*.

8.9.4 COMPLEX REASONING

An example of *complex reasoning* is exemplified in Figure 8-22 and suggests that a particular type of feature (non-through) that should not be split if it crosses another feature because of possible accessibility problems. It considers the volumetrical *cross* subjoint VI interaction between the features (thus it is +V) as well as their *labels* (thus it is +L).

```
(defrule cro_vi_1
(declare (salience 200))
(phase direct_related)
(object (is-a INTERACTION)
  (code cross)
  (name ?int)
  (master ?mstr)
  (n_slaves 1)
  (slaves ?slv)
  (status active))
(object (is-a FEATURE)
  (name ?mstr)
  (label ~hole&~slot_tru&~step&~gap)
  (nature ?n1)
  (status active))
(object (is-a FEATURE)
  (name ?slv)
  (volume rect)
  (nature ?n1) ; same natures 5-7-96
  (status active))
=>
(if (eq NO (Ask "Will resulting split features have ACCESSIBILITY ?" )
then (send ?int put-go_further NO)
      (send ?int put-status intentional )
else (send ?int put-status inactive)
      (send ?mstr ft_split_cross ?slv) )
)
```

Figure 8-22: A *Complex VDI Rule*.

8.9.4.1 TESTING THIN-WALLS

As discussed in Salmon (1997), a pure geometric reasoning approach for proximity (thin-wall) detection could be very demanding (requiring $(n^2 - n)/2$ Boolean intersection calculations) and cumbersome (requiring the use of a “Minkowski Sum” grow) for a precise detection. Bounding boxes were used instead.

Similarly, **FRIEND** uses the feature envelope and is helped by the availability of the feature interaction cases and the feature envelope face properties (*codes* and *projections*) which makes the **FRIEND** prediction of possible thin-wall problems an almost trivial activity.

To exemplify the *proximity* AOI, a thin-wall test rule using a fixed minimum wall thickness was established regardless of the machining method and material involved. Thin-wall cases were observed to originate from *adjoint* VI and *disjoint* VI cases and occur in two ways: feature-to-feature or feature-to-stock (see Figure 9-9):

- Feature-to-stock disjoint cases, where a thin-wall appears between two features that are *near* each other but not touching;
- Feature-to-feature adjoint cases, where a thin-wall appears between two features despite the fact that they are touching each other;
- Feature-to-stock adjoint cases, where a thin-wall appears when a feature touches the limits of the stock-material;
- Feature-to-feature disjoint cases, where a thin-wall appears when a feature is *near* the limits of the stock-material but not touching it;

In addition, a thin-wall AOI (TW) could be an *active*, *inactive* or *intentional* intent. An *active* TW intent means that there is a possible thin-wall problem yet to be resolved. An *inactive* TW intent means that there is not a thin-wall problem, the possibility of the existence of a thin-wall problem has been discarded already or it has already been resolved. An *intentional* TW intent means that there is not a thin-wall problem but the features are close enough to execute the respective rules or that the features are close enough and should be left as they are because of an unforeseen reason. Figure 8-23 presents an example of a disjoint thin-wall feature-to-feature AOI test rule.

```
(defrule tw_1
(phase approval_reasoning)
(object (is-a INTERACTION)
  (master ?mstr)
  (name ?int)
  (slaves ?slv)
  (type VI)
  (code disjoint)
  (status active))
(object (is-a FEATURE)
  (name ?mstr)
  (id ~STOCK)
  (volume rect)
  (status active))
(object (is-a FEATURE)
  (name ?slv)
  (id ~STOCK)
  (volume rect)
  (status active))
(test (<= (send (send ?mstr get-envelope) bb_distance (send ?slv get-envelope)) 5))
=>
(if (eq YES (Ask "Is there a THIN WALL between features ?" ))
then (send ?int put-code near)
      (send ?int put-go_further NO)
else (send ?int put-go_further NO)
      (send ?int put-code far))
)
```

Figure 8-23: A Thin-Wall AOI Test Rule.

For a more complete implementation of the thin-wall analysis, it would be necessary to take into account the feature types, their orientations, the selected machining processes and tools for each feature and the material involved.

8.10 INTENTS MANAGEMENT EXAMPLES

8.10.1 EXPERIENCE-BASED GUIDED ENRICHMENT

Figure 8-24 presents an example of a guided enrichment for the *co-radius* GDI. The rule automatically adds the INTENT between active features that are the result of splitting a feature.

```
(defrule en_cor_g
(phase enrichment)
(object (is-a INTENT)
  (code split_into)
  (slaves $?slvs)
  (status active))
(object (is-a FEATURE)
  (name ?slv_1)
  (label ?lbl)
  (volume cyl)
  (status active))
(object (is-a FEATURE)
  (name ?slv_2&~?slv_1)
  (label ?lbl)
  (volume cyl)
  (status active))
(test (member$ ?slv_1 ?slvs))
(test (member$ ?slv_2 ?slvs))
(not (object (is-a INTENT)
  (code co-radius)
  (master ?slv_1) (slaves ?slv_2)))
(not (object (is-a INTENT)
  (code co-radius)
  (master ?slv_2) (slaves ?slv_1)))
=>
(bind ?new (make-instance of INTENT
  (master ?slv_1) (slaves ?slv_2) (type StrGDI)))
(send ?new put-code co-radius)
(+ [REL] ?new)
(send ?new put-status active)
)
```

Figure 8-24: An Experience-based Guided GDI Enrichment.

Because of this it is not necessary to test their actual radii and it is considered that the *co-radius* GDI is an important relationship between these features to overcome the fact that they have been split. The four actions performed by this

rule are to create a structural GDI INTENT, instantiate the *co-radius* code, add the intent to the list of all RELATIONS and activate the new FbDI.

Figure 8-25 presents a guided enrichment for the *t_slot* AOI. It is executed when a *gap* feature (see feature taxonomy adopted in this work in section 1.5.2) is *nested@side* of another rectangular feature and no such INTENT (active or inactive) already exists.

```
(defrule en_aoi_ts
(phase enrichment)
(object (is-a FEATURE)
  (name ?f1)
  (volume rect)
  (label gap)
  (status active))
(object (is-a FEATURE)
  (name ?f2&~?f1)
  (volume rect)
  (label hole)
  (status active))
(object (is-a INTENT)
  (master ?f1)
  (slaves ?f2)
  (code nested@side)
  (status active))
(not (object (is-a INTENT)
  (master ?f1)
  (slaves ?f2)
  (code t_slot)))
=>
(bind ?new (make-instance of INTENT
  (master ?f1) (slaves ?f2) (type AOI)))
(send ?new put-code t_slot)

(if (eq YES (Ask "Can you work with a T_SLOT between master and slave"))
then
(send ?new put-status active)
else
(send ?new put-status inactive))

(+ [REL] ?new)
)
```

Figure 8-25: An AOI Enrichment Rule.

The AOI is created, instantiated and, according to the user's desire to acknowledge the FbDI, it is activated or inactivated. Therefore, if an inactive

AOI is created, it will prevent the system from asking again for this relation between the same features (unless the conditions change).

8.10.2 BLIND ENRICHMENT

Figure 8-26 presents a blind search for a *co-radius* GDI. It creates the INTENT for all pairs of cylindrical features that have equal radii where there is no (*active*, *inactive* or *intentional*) *co-radius* INTENT between them. No other condition is implied.

```
(defrule en_cor_b
(phase enrichment)
(object (is-a FEATURE)
(name ?mstr)
(volume cyl)
(status active))
(object (is-a FEATURE)
(name ?slv&~?mstr)
(volume cyl)
(status active))
(test (eq (send ?mstr get-radius) (send ?slv get-radius)))
(not (object (is-a INTENT)
(code co-radius)
(master ?mstr)
(slaves ?slv)))
(not (object (is-a INTENT)
(code co-radius)
(master ?slv)
(slaves ?mstr)))
=>
(bind ?new (make-instance of INTENT
(master ?mstr)
(slaves ?slv)
(type StrGDI)))
(send ?new put-code co-radius)
(+ [REL] ?new)

(if (eq YES (Ask "Create co-radius INTENT between features
?"))
then; nothing to do, the default status is active
else (send ?new put-status inactive))
)
```

Figure 8-26: Blind *Co-radius* GDI Enrichment.

8.10.3 VERIFICATION

Figure 8-27 presents a verification for the *co-radius* GDI. It simply checks the `radius` of features in the `INTENT`. It should either inactivate the GDI or change one of the features involved to make it comply with the conditions. Function *not_yet* means that the option has not yet been implemented.

```
(defrule ve_cor
  (phase enrichment)
  (object (is-a FEATURE)
    (name ?mstr)
    (volume cyl))
  (object (is-a FEATURE)
    (name ?slv)
    (volume cyl))
  (test (neq (send ?mstr get-radius) (send ?slv get-radius)))
  (object (is-a INTENT)
    (name ?int)
    (code co-radius)
    (master ?mstr)
    (slaves ?slv)
    (status active|intentional))
  =>
  (if (eq YES (Ask "Delete co-radius GDI between master and slave? "
    then
      (send ?int put-status inactive)
    else
      (not_yet "Should suggest change radius of master or slave")
  ))
```

Figure 8-27: A *Co-radius* GDI Verification.

Figure 8-28 presents a verification statement for the *t_slot* AOI. It tests the status and type of the features involved as well as the required *nested@side* INTENT between them. It is implemented in an alternative, but equivalent, way to the outline given in Figure 7-4 and Figure 7-5: It should be noticed that a Boolean test $\sim(\text{Cond}_1 \text{ AND } \text{Cond}_2 \text{ AND } \dots)$ is equal to $(\sim\text{Cond}_1 \text{ OR } \sim\text{Cond}_2 \text{ OR } \dots)$. If the *t_slot* is to be removed it is inactivated, otherwise it is made intentional because it can not be considered active according to how it was defined here.

```
(defrule ve_aoi_ts
  (phase enrichment)
  (object (is-a INTENT)
    (name ?intent)
    (master ?f1)
    (slaves ?f2)
    (code t_slot)
    (status active))
  (not (and
    (object (is-a FEATURE)
      (name ?f1)
      (volume rect)
      (label gap)
      (status active))
    (object (is-a FEATURE)
      (name ?f2)
      (volume rect)
      (label hole)
      (status active))
    (object (is-a INTENT)
      (master ?f1)
      (slaves ?f2)
      (code nested@side)
      (status active))))))
=>
  (if (eq YES (Ask "Delete T_SLOT between master and slave ? " ))
    then
      (send ?intent put-status inactive)
    else
      (send ?intent put-status intentional)
      ; should edit feature1 or feature2
      )
  )
)
```

Figure 8-28: An AOI Verification Rule.

8.10.4 INHERITANCE-BASED GUIDED ENRICHMENT

Figure 8-29 presents an enrichment based on the fact that if a feature was created from the application of the split *revalidation* operation on another feature, the INTENTs assigned to the original feature are automatically inherited by the newly generated feature. It was found that most INTENTs conditioned by disjoint VI interaction cases can always be inherited. However, not all INTENTs conditioned by adjoint VI cases can always be inherited.

```
(defrule inherit_splt_act
  (phase enrichment)
  (object (is-a INTENT)
    (master ?mstr)
    (code split_into)
    (slaves $? ?slv $?)
    (status active))
  (object (is-a INTENT)
    (name ?int)
    (master ?mstr)
    (code ?cd&obsoleted_bydeleted_by)
    (type ?tp)
    (n_slaves ?nslvs)
    (slaves $?the_other_slaves)
    (status inactive))
  (object (is-a FEATURE)
    (name ?slv)
    (status active!intentional))
  (not (object (is-a INTENT)
    (master ?slv)
    (code ?cd)
    (type ?tp)
    (slaves $?the_other_slaves)))
  =>
  ; The original intent is removed
  (send ?int put-status intentional)

  (bind ?new (make-instance of INTENT
    (type ?tp) (master ?slv) (slaves ?the_other_slaves) (n_slaves ?nslvs)))
  (send ?new put-code ?cd)
  (+ [REL] ?new)

  (if (eq YES (Ask "Inherent INTENT form master ?"))
    then ; nothing to do, default status is active
    else (send ?new put-status inactive))
  )
```

Figure 8-29: An Inheritance Enrichment Rule Example.

8.11 FINAL REMARKS ON THE IMPLEMENTATION

This research was conducted in two main implementation phases:

- In the initial phase, the *conceptual* feature-based representation validation problem was identified (see section 1.2) and its origins (see chapter 5 and section 3.5) were analysed.

This led to the establishment of a validation framework (see section 3.3) and some revalidation operations (section 6.2.3.3). Subsequently, a priority organisation implied by the reasoning was obtained (section 7.2).

- In the second phase, the validation framework and the VDI's were thought to be suitable to be extended towards an intent-driven reasoning system where other types of FbDI's were to be included (see section 4.3).

However, a methodology to identify the elements of this wider system became necessary. Because a classification for features existed and similarly for feature interaction cases, this was considered an important aspect of the validation framework and as a possible approach.

It was decided that a process similar to feature elicitation (see section 1.5.2) was to be used towards the other elements (see section 4.2 and 6.1) which then helped identify and clarify FbDI's (see section 4.4) and operations (section 6.3). In addition, the ways of reasoning were extended to accommodate the other FbDI types (section 7.3).

8.12 SUMMARY

The implementation of a prototype system, called **FRIEND**, has allowed the verification of the elements studied here (i.e. feature interactions, feature-based designer's intents, reasoning priority, intent management and feature-based operations) to compose a working DbF system.

The interaction identification methodology has been applied using the feature envelope (bounding box) instead of the actual volume. Invalidity tests were implemented using rules in a knowledge-based system. Rules were typified and exemplified.

The visualisation was carried out by a commercial CAD system that communicates with the reasoning system.

9. TEST CASES

This chapter presents some feature-based part models as test cases for FRIEND. Some of these models have been used in the literature as test cases for feature-based modelling system implementations. It aims to show that the prototype system is able to represent and reason with components which have been modelled by and used to test the capabilities of other feature-based modellers.

9.1 INTRODUCTION

This chapter presents some test parts that are adaptations of parts published in the literature and modelled using feature-based modellers. They are adaptations of the original parts because:

- Dimensions are frequently not specified for the parts;
- The feature taxonomy used to describe the part could be different from that used by **FRIEND**;
- Some invalid situations were deliberately introduced in the part definitions to observe how **FRIEND** would respond to them;
- Some features implemented in other systems have not been implemented in the prototype system;

- Some geometric configurations have been simplified in order to save computing time.

9.2 STANDARD ORIENTATION

Figure 9-1 presents most of the feature types implemented in **FRIEND**. They are placed in their standard orientation, i.e. the features are translated but not rotated relative to their defining template. This is a different way of presenting **FRIEND**'s feature taxonomy depicted in Figure 8-19.

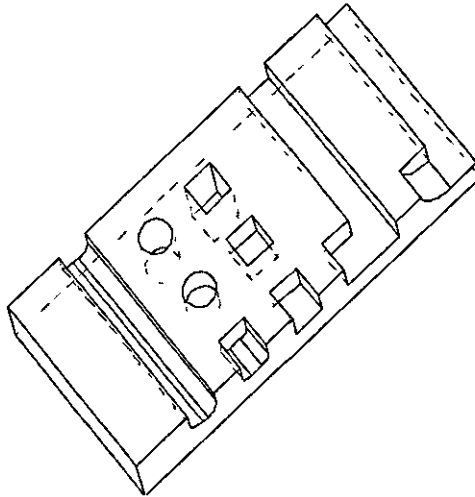


Figure 9-1: Features in Standard Orientation.

The model is defined via a part description file (".prt") which is a simple way of describing all features that represent a model. This facility avoids the task of redefining the model feature by feature every time an analysis is to be performed. The '.prt' file for the part in Figure 9-1 appears in Figure 9-2. The ".prt" file defines:

- the total number of features in the part (where the stock-material is considered a *satellite* feature which has been in Figure 1-6);
- their assigned names (which has nothing to do with their *labels*);

- their *volume* type (*rect* or *cyl*);
- a radius to be used if the feature is cylindrical;
- two rotations to identify the orientation of the feature on the part;
- two points to specify the envelope (bounding box): the minimum (bottom-left corner) and maximum (top-right corner) vertices of the bounding box in a left-handed coordinate system.

```
12
STOCK, satellite, rect, 0
orig, orig
-5 0 0 60 10 30
A, slot_tru, cyl, 2
orig, orig
5 5 0 10 10 30
B, slot, cyl, 1.5
orig, orig
15 5 0 20 10 5
C, pocket, cyl, 2.5
orig, orig
15 5 10 20 10 15
D, hole, cyl, 2.5
orig, orig
15 0 20 20 10 25
E, slot, rect, 0
orig, orig
25 5 0 30 10 5
F, pocket, rect, 0
orig, orig
25 5 10 30 10 15
G, hole, rect, 0
orig, orig
25 0 20 30 10 25
H, slot_tru, rect, 0
orig, orig
40 5 0 45 10 30
I, notch, rect, 0
orig, orig
35 5 0 40 10 5
J, step, rect, 0
orig, orig
55 5 0 60 10 30
K, notch, cyl, 5
orig, orig
50 5 0 55 10 5
```

Figure 9-2: Part Description File Example.

It should be noted that, because the features have faces with determined properties that go beyond simply describing the primitive volume (such as Top and Bottom, which could be thought of as a way of describing an upright orientation), two rotations are required and sufficient to obtain any orientation in a 3D environment. These rotations are limited though to a group which always produces a feature parallel to one of the Euclidean axes.

The rotations in Figure 9-2 are set as “orig, orig”, which means that their orientation have been left as the default. The features are called A, B, C, E, etc.

The stock-material is considered to be a rectangular *satellite* feature of positive *nature* and should be defined as the first feature in the model which contains the remaining negative features. In addition, similarly to other work (Gindy89, Gao93), “blind” *holes* are classified as *pocket* features with a round or quadrangular profile.

9.3 LABELLING

Figure 9-3 presents a part from Martino and Giannini (1994a) where the *labelling* problem is highlighted. It is shown that the addition of a feature into a model could change all the existing features in the model.

Figure 9-3(a) presents the original part containing an upside down *pocket* (elsewhere called a non-through or blind *hole*) and a quadrangular *hole* (also called a through *hole*). The addition of a *step* feature renders the existing *hole* and *pocket* features invalid. Figure 9-3(b) presents the final part comprised of the incoming *step*, a *slot_tru* (originally the *hole*) and another *hole* (formerly the *pocket*).

Figure 9-4 presents the description of the part in Figure 9-3(b) before and after the validation reasoning.

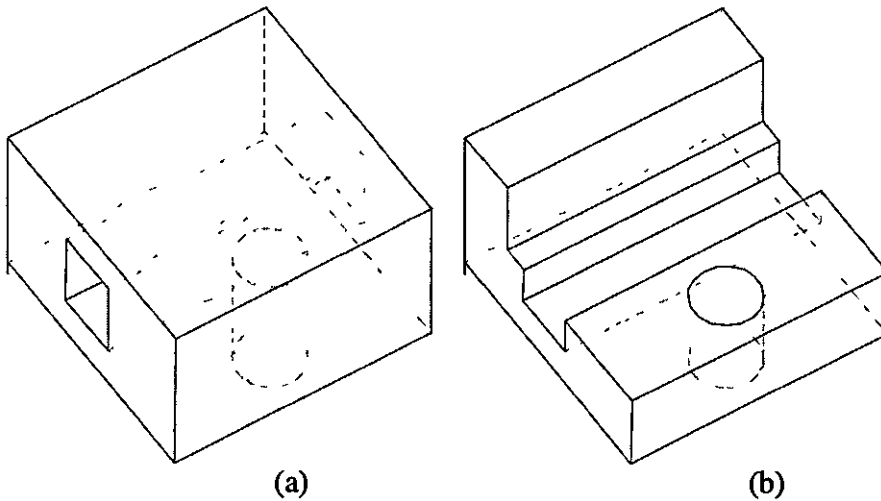


Figure 9-3: Martino and Giannini's Part.

In this example, the major differences between the valid and invalid parts are (see bold text in Figure 9-4):

- the HOLE-TRU feature, labelled as *hole* initially, receives the *invalid* status and is split into HOLE-TRU_1 and HOLE-TRU_2. HOLE-TRU_1 is redundant to the volume of the *step* and therefore it is made obsolete and thus receives the *intentional* status. HOLE-TRU_2 is the remaining part that really affects the stock and thus receives the validated *slot_tru* label and an *active/valid* status. It should be noted that both envelopes are derived from the HOLE-TRU feature.
- Similarly, the BLIND-HOLE feature, labelled correctly at the beginning as a *pocket*, is split into invalid and valid parts. The valid part, BLIND-HOLE_2, is validated as a *hole* feature and receives the *active* status.
- The STEP feature, labelled *step*, has indeed been found to be a *step* feature but **FRIEND** corrects its orientation (compare the orientation of the step features in Figure 9-3(b) with that presented in Figure 9-1).

Both HOLE-TRU_1 and BLIND-HOLE_1 become intentional features because their volumetric intention can reappear if the STEP feature is deleted.

Before and After the Validation Reasoning
4 STOCK satellite, rect, 0 orig, orig 0 0 0 20 10 20 HOLE-TRU hole, rect, 0 orig, rz90 0 2.5 8 20 7.5 13 BLIND-HOLE pocket, cyl, 2 5 orig, rz180 7.5 0 1.5 12.5 7.5 6.5 STEP step, rect, 0 orig, orig 0 5 0 20 10 15	(1): STOCK(gen1),BLUE (satellite, rect, active), BBox: (1:orig,2:orig) P = 0 0 0.0 0.0 P = 20 0 10 0 20 0 (2): HOLE-TRU(gen7),_ (hole, rect, inactive), BBox: (1:orig,2:rz90) P = 0.0 2.5 8.0 P = 20 0 7.5 13.0 (3): BLIND-HOLE(gen16),_ (pocket, cyl, inactive), BBox: (1:orig,2:rz180) P = 7.5 0.0 1.5 P = 12.5 7.5 6.5 (4): STEP(gen25),GREEN (step, rect, active), BBox: (1:orig,2:ry90) P = 0.0 5.0 0 0 P = 20.0 10.0 15.0 (5): HOLE-TRU_1(gen58),_ (hole, rect, intentional), BBox: (1:orig,2:rz90) P = 0.0 5.0 8.0 P = 20.0 7.5 13.0 (6): HOLE-TRU_2(gen64),RED (slot_tru, rect, active), Bbox: (1:orig,2:ry90) P = 0.0 2.5 8.0 P = 20.0 5.0 13.0 (7): BLIND-HOLE_1(gen94),_ (pocket, cyl, intentional), BBox: (1:orig,2:rz180) P = 7.5 5.0 1.5 P = 12.5 7 5 6 5 (8): BLIND-HOLE_2(gen100),YELLOW (hole, cyl, active), BBox: (1:orig,2:orig) P = 7 5 0 0 1.5 P = 12.5 5.0 6.5

Figure 9-4: Martino and Giannini's Part Description.

9.4 VALID PART DESCRIPTION

Figure 9-5 reproduces the part presented in Figure 1-9 before (a) and after (b) the validation reasoning performed by **FRIEND**. Figure 9-6 shows the corresponding part description file. The left-hand side of the figure shows the non-validated representation and the right-hand side shows **FRIEND**'s output. The name of the feature is usually maintained from the ".prt" file unless the feature is split or merged with another in which case it will receive a numeric addendum (see features number 5 and 6 in the right-hand side of Figure 9-6) or the "+" sign (see feature number 8 in Figure 9-6) to indicate the original features, respectively.

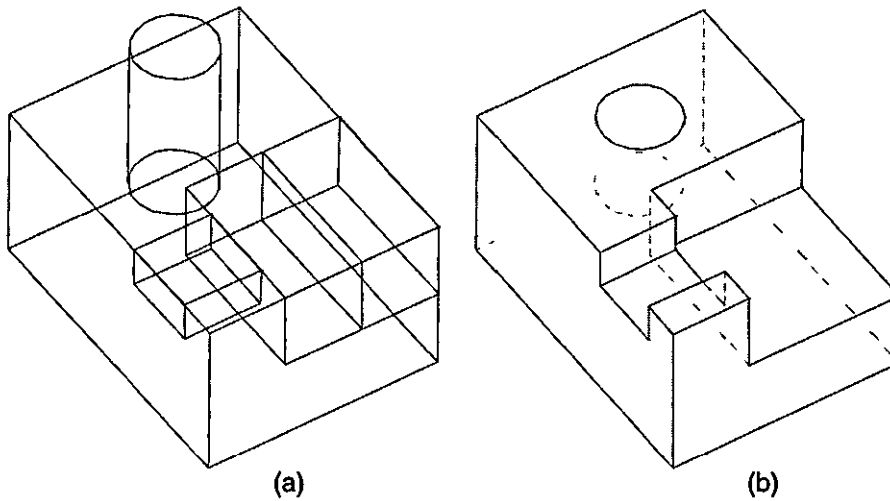


Figure 9-5: An Example Part Before (a) and After (b) Validation.

The output produced by **FRIEND** lists all features in the model. This list can also include *invalid/inactive* and *intentional* features in addition to the *valid/active* ones. The output gives the following information:

- the name of the feature;
- an internal variable (e.g. *gen60*) and a colour (only if it is a *valid* feature and should be visualised via the **FRIEND-VIEW** module);
- the *label*;
- the volume type (*rect* or *cyl*);
- the status (*valid*, *invalid* or *intentional*);
- the validated envelope (**Bbox**) orientation represented by two rotations;
- two points that specify the size and position of the envelope (minimum and maximum points);

Before and After the Validation Reasoning
5 STOCK satellite, rect, 0 orig, orig 0 0 0 15 10 20	(1): STOCK(gen1),BLUE (satellite, rect, active), BBox: (1:orig,2:orig) P = 0.0 0.0 0.0 P = 15.0 10 0 20.0
Entalhe notch, rect, 0 orig, orig 10 5 0 15 10 10	(2): Entalhe(gen7),_ (notch, rect, inactive), BBox: (1:orig,2:orig) P = 10. 5. 0. P = 15. 10. 10.
RasgoNaoPassante hole, rect, 0 orig, orig 0 7.5 2.5 5 10 7.5	(3): RasgoNaoPassante(gen17),GREEN (slot_tru, rect, active), BBox: (1:orig,2:ry90) P = 0 7.5 2.5 P = 5. 10. 7.5
FuroNaoPassante slot_tru, cyl, 2.5 orig, orig 5 5 12.5 10 15 17.5	(4): FuroNaoPassante(gen51),_ (pocket, cyl, inactive), BBox: (1:orig,2:orig) P = 5. 5. 12.5 P = 10. 15. 17.5
RasgoQueAlteraTudo slot, rect, 0 orig, orig 5 5 0 10 10 10	(5): FuroNaoPassante_1(gen60),RED (pocket, cyl, active), BBox: (1:orig,2:orig) P = 5. 5. 12.5 P = 10. 10. 17.5
	(6): FuroNaoPassante_2(gen66),_ (pocket, cyl, inactive), BBox: (1:orig,2:orig) P = 5. 10. 12.5 P = 10. 15. 17.5
	(7): RasgoQueAlteraTudo(gen82),_ (notch, rect, inactive), BBox: (1:orig,2:orig) P = 5. 5. 0. P = 10. 10. 10.
	(8): Rasgo&Entae+(gen126),YELLOW (notch, rect, active), BBox: (1:orig,2:orig) P = 5. 5. 0. P = 15. 10. 10.

Figure 9-6: Non-validated and Validated Model Description.

In addition, FRIEND was able to merge adjacent features to compose a “Rasgo&Entae+” feature, labelled *notch*, at the same time that it split the “FuroNaoPassante” feature, labelled *hole*, and discarded (made inactive) the obsolete part “FuroNaoPassante_2”. FRIEND also corrects the label of the resulting “FuroNaoPassante_1” and calls it a *pocket* feature. These features are highlighted in the respective descriptions. Both these reasonings are related to the *fittability* VDI (see section 3.5.2.2) where the features had parameters too small or too large, respectively.

Feature RasgoNaoPassante has been incorrectly defined as a slot feature and it appears corrected as a slot_tru feature (see feature 3 in the right-hand side of Figure 9-6). This is a typical example of the result of a simply labelling reasoning presented in section 8.9.3 (Figure 8-21).

As a result of the reasoning, some features were made *inactive* in order to give rise to a more accurate representation of the model using *active* features.

9.5 MORPHOLOGICAL REASONING TEST

Figure 9-7 presents a part similar to Figure 3-10 where a complete conceptual morphological validation process is carried out. Figure 9-7(a) shows the part with the original volumes of the features while Figure 9-7(b) shows the output after the application of associated Boolean operations.

FRIEND is able to discard part of the cylindrical *slot* outside the stock-material and the part overlapping the other *slot* feature.

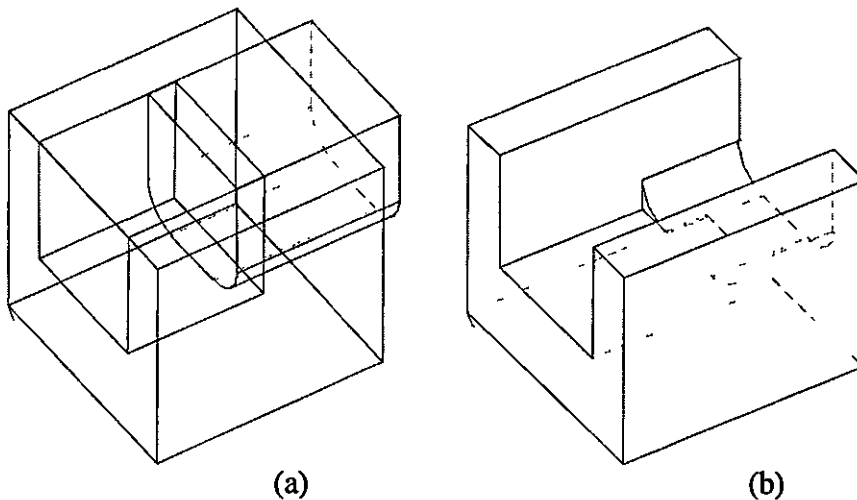


Figure 9-7: Morphological Validation Reasoning.

In addition, the first added feature (“EnteredFeature”) is incorrectly defined as a *pocket* instead of a *slot*, which appears corrected in the output listing (Figure 9-8). However, **FRIEND** does not merge the resulting slots (as happens in

Figure 3-10) because the resulting features have different radii. Nevertheless, the second cylindrical *slot* is redefined as *slot_tru*.

Figure 9-8 presents the description of the part in Figure 9-7 before and after the conceptual morphological validation reasoning.

Before and After the Validation Reasoning
3 STOCK satellite, rect, 0 orig, orig 0 0 0 25 25 25	(1). STOCK(gen846),BLUE (satellite, rect, active), BBox: (1:orig,2:orig) P = 0 0 0.0 0.0 P = 25.0 25.0 25.0
EnteredFeature pocket , rect, 0 orig, rz270 0 10 5 15 25 20	(2). EnteredFeature(gen852),GREEN (slot, rect, active), BBox: (1:orig,2:ry90) P = 0 0 10.0 5.0 P = 15.0 25.0 20.0
EnteringFeature slot, cyl, 4 orig, ry90 12 10 5 25 25 20	(3). EnteringFeature(gen891),_ (slot, cyl, inactive), BBox: (1:orig,2:ry90) P = 12.0 10.0 5.0 P = 30.0 25.0 20.0
	(4). EnteringFeature_1(gen899),_ (slot, cyl, inactive), BBox: (1:orig,2:ry270) P = 12 0 10.0 5 0 P = 25 0 25 0 20 0
	(5). EnteringFeature_2(gen905),_ (slot, cyl, inactive), BBox: (1:orig,2:ry90) P = 25 0 10 0 5 0 P = 30.0 25.0 20.0
	(6). EnteringFeature_1_1(gen932),_ (slot, cyl, intentional), BBox: (1:orig,2:ry270) P = 12 0 10 0 5 0 P = 15.0 25.0 20.0
	(7). EnteringFeature_1_2(gen938),RED (slot_tru, cyl, active), BBox: (1:orig,2:ry90) P = 15.0 10.0 5.0 P = 25.0 25.0 20.0

Figure 9-8: Part Description Before and After Validation.

Because part of the “EnteringFeature” has a *redundant* VDI with the “EnteredFeature” the corresponding feature after the split revalidation operation (“EnteredFeature_1_1”) receives the *intentional* status. This means that if the former “EnteredFeature” is deleted from the model, “EnteredFeature_1_1” can become *active* again.

9.6 THIN WALL TEST CASES

Figure 9-9 shows four example parts produced to demonstrate how **FRIEND** identifies *proximity* AOF's (thin-walls). The figure suggests that thin-wall reasoning can be built upon feature interaction cases:

- feature-to-feature adjoint cases are exemplified by a part where a curved face of a *step* touches the bottom cylindrical face of a *hole* feature;
- feature-to-feature disjoint cases are exemplified by a part where a *through slot* is too close to a *step* feature;
- feature-to-STOCK adjoint cases are exemplified by a part where a curved face of a *step* feature occurs at the limits of the stock material, and;
- feature-to-STOCK disjoint cases are exemplified by a part where *hole* features are too close to the limits of the stock material.

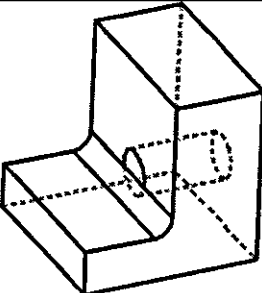
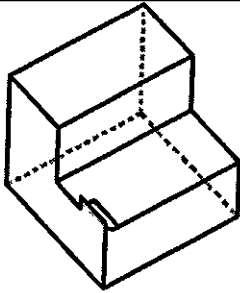
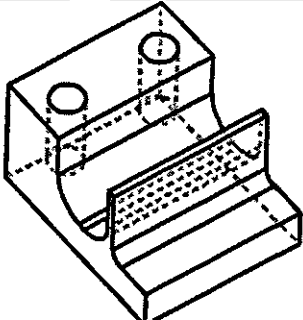
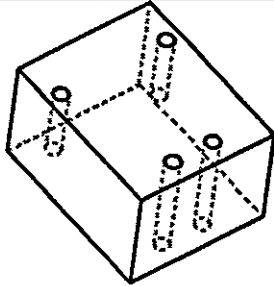
	Feature-to-Feature	Feature-to-STOCK
Adjoint Cases		
Disjoint Cases		

Figure 9-9: Thin-Wall Cases.

Figure 9-10 presents the validated output produced by **FRIEND** for the part exemplified in the bottom left corner of Figure 9-9, which corresponds to a feature-to-feature disjoint thin-wall case. The rule for this case was presented in Figure 8-23. The volumetric (**VI**) and boundary (**BI**) interactions that were obtained from the model and used in the reasoning of *proximity* AOI are shown in Figure 9-11

```
(1): STOCK(gen1),BLUE (satellite, rect, active), BBox: (1:orig,2:orig)
P = 0.0 0.0 0 0 P = 20.0 15.0 25.0

(2): green(gen7),GREEN (slot_tru, cyl, active), BBox: (1:orig,2:ry90)
P = 0.0 5.0 7.0 P = 20.0 15.0 16.0

(3): red(gen23),RED (step, cyl, active), BBox: (1:orig,2:ry90)
P = 0.0 5.0 0.0 P = 20 0 15.0 6 0

(4): yellow(gen35),YELLOW (pocket, cyl, active), BBox: (1:orig,2:orig)
P = 2.0 5.0 18 0 P = 6 0 15.0 22 0

(5). pink(gen51),PINK (pocket, cyl, active), BBox: (1:orig,2:orig)
P = 14 0 5 0 18 0 P = 18 0 15.0 22.0
```

Figure 9-10: Description of a Part containing Thin-Walls.

```
(red,RED) -> near -> (1): (green,GREEN)
(yellow,YELLOW) -> near -> (1): (green,GREEN)
(yellow,YELLOW) -> disjoint -> (1): (red,RED)
(pink,PINK) -> near -> (1): (green,GREEN)
(pink,PINK) -> disjoint -> (1): (red,RED)
(pink,PINK) -> disjoint -> (1): (yellow,YELLOW)
(green,GREEN) -> limited -> (1): (STOCK,BLUE)
(red,RED) -> limited -> (1): (STOCK,BLUE)
(yellow,YELLOW) -> limited -> (1): (STOCK,BLUE)
(pink,PINK) -> limited -> (1): (STOCK,BLUE)
```

Figure 9-11: Volumetric and Boundary Interactions.

It is considered that the following thin-wall cases happen in this part: between the *step* called “red” and the *slot_tru* feature called “green”; between both *pocket* features, called “yellow” and “pink” and the *slot_tru* feature called “green”. The existence of this type of *proximity* AOI can be tested directly from the disjoint volumetric interaction (**VI**) cases, which can thus be

confirmed or not, generating the *near* or *far* disjoint VI cases (highlighted in Figure 9-11, see also section 8.9.4.1).

9.7 CHANG'S PART

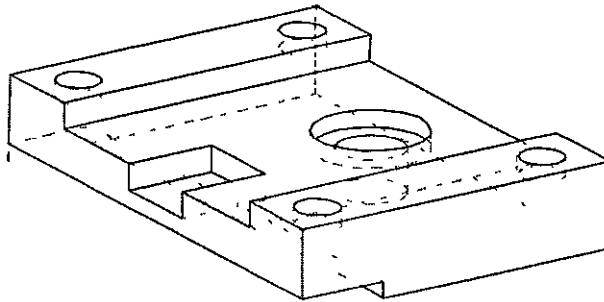


Figure 9-12: Chang's Test Part.

Chang (1990:208) studied the problems associated with expert process planning for manufacturing. Figure 9-12 reproduces a test part used to discuss the problems and reasonings related to the generation of automatic process plans. Figure 9-13 presents the validated output for this test part.

```
(1): STOCK(gen585),BLUE (satellite, rect, active), BBox: (1:orig,2:orig)
P = 0.0 0.0 0.0 0.0 P = 15.0 10.0 22.0
(2): CentreSlot-F1(gen591),GREEN (slot_tru, rect, active), BBox: (1:orig,2:ry90)
P = 0 0 7.0 4 0 P = 15 0 10.0 18.0
(3): MidHole-F12(gen637),RED (pocket, cyl, active), BBox: (1:orig,2:orig)
P = 8.0 5 0 8 5 P = 13 0 7.0 13 5
(4): MidHole-F13(gen70),YELLOW (hole, cyl, active), BBox: (1:orig,2:orig)
P = 9 0 0.0 9 5 P = 12 0 5.0 12.5
(5): EnterSlot-F14(gen682),PINK (slot, rect, active), BBox: (1:orig,2:ry90)
P = 0 0 4.0 9 0 P = 4.0 7.0 13 0
(6): UnderStep-F15(gen719),RED (step, rect, active), BBox: (1:orig,2:rz180)
P = 0.0 0.0 0.0 P = 4.0 2 0 22 0
(7): Hole1-F10(gen771),LIGHTBLUE (pocket, cyl, active), BBox: (1:orig,2:orig)
P = 1.0 8 0 1.0 P = 3 0 10.0 3.0
(8): Hole2-F9(gen790),BLACK (pocket, cyl, active), BBox: (1:orig,2:orig)
P = 12.0 8.0 1.0 P = 14.0 10.0 3.0
(9): Hole3-F7(gen814),GRAY (pocket, cyl, active), BBox: (1:orig,2:orig)
P = 12 0 8 0 19 0 P = 14.0 10.0 21.0
(10): Hole4-F8(gen843),RED (pocket, cyl, active), BBox: (1:orig,2:orig)
P = 1 0 8.0 19 0 P = 3 0 10.0 21.0
```

Figure 9-13: Validated Output for Chang's Test Part.

One strategy adopted by Chang (1990) was to identify “clusters” of features that share the same tool and/or tool access direction. This information is used to reason about setup planning. A hierarchical graph that identifies various types of precedence (such as *structural* precedence due to process geometry constraints and *loose* precedence due to good manufacturing practice) is considered for reasoning about precedence planning.

```
(MidHole-F12,RED) -> nested@bot -> (1): (CentreSlot-F1,GREEN)
(MidHole-F13,YELLOW) -> nested@bot -> (1): (MidHole-F12,RED)
(MidHole-F13,YELLOW) -> parallel -> (1): (MidHole-F12,RED)
(MidHole-F13,YELLOW) -> concentric -> (1): (MidHole-F12,RED)
(EnterSlot-F14,PINK) -> nested@bot -> (1): (CentreSlot-F1,GREEN)
(EnterSlot-F14,PINK) -> parallel -> (1): (CentreSlot-F1,GREEN)
(Hole1-F10,LIGHTBLUE) -> parallel -> (1): (MidHole-F13,YELLOW)
(Hole1-F10,LIGHTBLUE) -> parallel -> (1): (MidHole-F12,RED)
(Hole2-F9,BLACK) -> co-radius -> (1): (Hole1-F10,LIGHTBLUE)
(Hole2-F9,BLACK) -> parallel -> (1): (Hole1-F10,LIGHTBLUE)
(Hole2-F9,BLACK) -> parallel -> (1): (MidHole-F13,YELLOW)
(Hole2-F9,BLACK) -> parallel -> (1): (MidHole-F12,RED)
(Hole3-F7,GRAY) -> co-radius -> (1): (Hole2-F9,BLACK)
(Hole3-F7,GRAY) -> co-radius -> (1): (Hole1-F10,LIGHTBLUE)
(Hole3-F7,GRAY) -> parallel -> (1): (Hole2-F9,BLACK)
(Hole3-F7,GRAY) -> parallel -> (1): (Hole1-F10,LIGHTBLUE)
(Hole3-F7,GRAY) -> parallel -> (1): (MidHole-F13,YELLOW)
(Hole3-F7,GRAY) -> parallel -> (1): (MidHole-F12,RED)
(Hole4-F8,RED) -> co-radius -> (1): (Hole3-F7,GRAY)
(Hole4-F8,RED) -> co-radius -> (1): (Hole2-F9,BLACK)
(Hole4-F8,RED) -> co-radius -> (1): (Hole1-F10,LIGHTBLUE)
(Hole4-F8,RED) -> parallel -> (1): (Hole3-F7,GRAY)
(Hole4-F8,RED) -> parallel -> (1): (Hole2-F9,BLACK)
(Hole4-F8,RED) -> parallel -> (1): (Hole1-F10,LIGHTBLUE)
(Hole4-F8,RED) -> parallel -> (1): (MidHole-F13,YELLOW)
(Hole4-F8,RED) -> parallel -> (1): (MidHole-F12,RED)

(MidHole-F13,YELLOW) -> c_bore -> (1): (MidHole-F12,RED)
(Hole1-F10,LIGHTBLUE) -> coEAD -> (1): (MidHole-F12,RED)
(Hole2-F9,BLACK) -> coEAD -> (1): (Hole1-F10,LIGHTBLUE)
(Hole2-F9,BLACK) -> coEAD -> (1): (MidHole-F12,RED)
(Hole3-F7,GRAY) -> coEAD -> (1): (Hole2-F9,BLACK)
(Hole3-F7,GRAY) -> coEAD -> (1): (Hole1-F10,LIGHTBLUE)
(Hole3-F7,GRAY) -> coEAD -> (1): (MidHole-F12,RED)
(Hole4-F8,RED) -> coEAD -> (1): (Hole3-F7,GRAY)
(Hole4-F8,RED) -> coEAD -> (1): (Hole2-F9,BLACK)
(Hole4-F8,RED) -> coEAD -> (1): (Hole1-F10,LIGHTBLUE)
(Hole4-F8,RED) -> coEAD -> (1): (MidHole-F12,RED)
```

Figure 9-14: Designer’s Intents to Assist Precedence and Setup Planning.

Although generating plans is not **FRIEND**'s major concern, it gathers valuable information during the design process that can be readily used for similar clustering and hierarchical reasoning. For instance, the various types of GDP's and AOI's presented in Figure 9-14 obtained while modelling the part are examples of such valuable information. In addition, **FRIEND** recognises the existence of a *compound* AOI and assigns the appropriate *counter_bore* intent between the *hole* features in the test part.

9.8 A LOST INTENTION?

Perng and Chang (Perng97b) studied the problems associated with editing a feature-based model. Figure 9-16 gives the validated output description of the part in the Figure 9-15(b). It should be noted (see bold text in the figure) that some features have an *inactive* status (such as *Hole1*, which was split due to a *cross* interaction with USlotTop) while others have an *intentional* status (such as *Hole1_1*, *Hole1_2* and *Hole1_3*, which had their volumes *obsoleted_by* other feature).

The major concern was the efficient updating of the solid modelling (both B-rep and CSG) representations. The conceptual validation problem for the part shown in Figure 9-15(a) arose where the enlargement of the T-slot top part led to the *Hole1* feature vanishing. Besides asking "how can the system derive the modified B-rep efficiently", it was also necessary to ask "How shall the vanished *Hole1* be dealt with?".

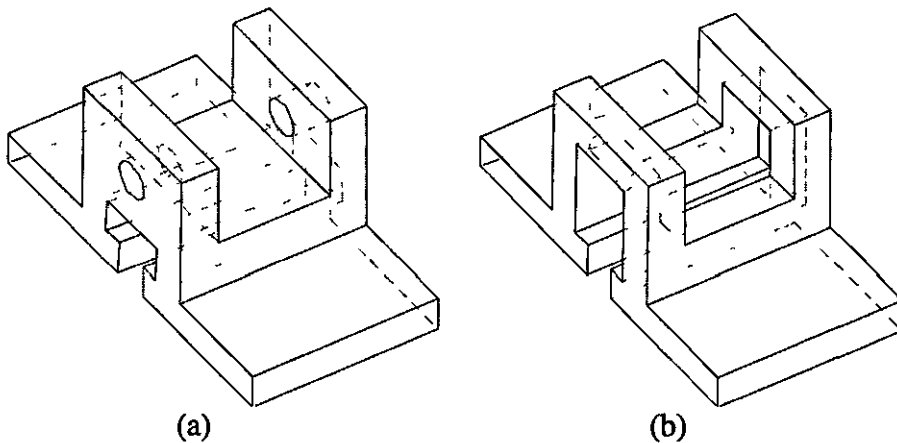


Figure 9-15: The Vanished *Hole* Feature.

FRIEND deals with the problem posed by Perng and Chang in the following way: Every time a feature *volume* becomes contained within another feature *volume*, the former is made obsolete and receives an *intentional* status. This happens for the part of the *hole* feature that is crossed by the *USlot-Top* as well as for both remaining *hole* features that became contained within the top of the *T-Slot* feature. This means that if the *TSlotTop* is subsequently removed or reduced in size the *hole* feature will reappear in the model. It should be noted that although **FRIEND** does not provide a *T-Slot* feature, it recognises it using the rule given in Figure 8-25 and assigns the appropriate feature-based designer's intent (the *t_slot* AOD) to the model.

Figure 9-17 presents the FbDI's gathered from the resulting part discussed above. The *split_into* and the *obsoleted_by* VDI's mentioned above are highlighted in the figure. Some of the FbDI's have an *intentional* status because the relationship has only one *active* feature.

Note that *split_into* VDI has an *inactive* status according to the "act_int_dead_feat" intent management rule described in section 7.4.3.

(1): STOCK(gen1),BLUE (satellite, rect, active), BBox: (1.orig,2.orig)
P = 0 0 0.0 0.0 P = 10 0 10.0 20.0

(2): Step1(gen7),GREEN (step, rect, active), BBox: (1.orig,2.ry90)
P = 0 0 2.0 0.0 P = 10 0 10.0 6.0

(3): Step2(gen36),RED (step, rect, active), BBox: (1.orig,2.ry270)
P = 0.0 2.0 14.0 P = 10.0 10.0 20 0

(4): USlot-Top(gen67),YELLOW (slot_tru, rect, active), BBox: (1.orig,2.orig)
P = 2.0 5 0 6.0 P = 8.0 10.0 14.0

(5): Hole1(gen94),_ (hole, cyl, inactive), BBox: (1.orig,2.rz90)
P = 0.0 6.0 9.0 P = 10 0 8 0 11 0

(6): Hole1_1(gen106),_ (hole, cyl, intentional), BBox: (1.orig,2.rz90)
P = 0.0 6.0 9.0 P = 2.0 8.0 11.0

(7): Hole1_2(gen112),_ (hole, cyl, intentional), BBox: (1.orig,2.rz90)
P = 2.0 6.0 9.0 P = 8.0 8.0 11.0

(8): Hole1_3(gen118),_ (hole, cyl, intentional), BBox: (1.orig,2.rz90)
P = 8.0 6.0 9.0 P = 10.0 8.0 11.0

(9): TSlotUnder(gen156),PINK (gap, rect, active), BBox: (1.orig,2.ry90)
P = 0.0 0 0 9.0 P = 10 0 2 0 11.0

(10): TSlotTop(gen230),RED (hole, rect, active), BBox: (1.orig,2 rz90)
P = 0.0 2.0 8.0 P = 10 0 8 0 12.0

Figure 9-16: Description of Perng and Chang's Part.


```

INTENT HieGDI(gen398) Go: YES, active
(USlot-Top,YELLOW) -> nested@bot -> (1): (Step1,GREEN)

INTENT HieGDI(gen399) Go: YES, active
(USlot-Top,YELLOW) -> nested@bot -> (1): (Step2,RED)

INTENT VDI(gen431) Go: NO, inactive
(Hole1,_) -> split_into -> (2): (Hole1_1,_) (Hole1_2,_) (Hole1_3,_)

INTENT VDI(gen448) Go: YES, intentional
(Hole1_2,_) -> obsoleted_by -> (1): (USlot-Top,YELLOW)

INTENT HieGDI(gen453) Go: YES, intentional
(Hole1_1,_) -> nested@side -> (1): (USlot-Top,YELLOW)

INTENT HieGDI(gen454) Go: YES, intentional
(Hole1_3,_) -> nested@side -> (1): (USlot-Top,YELLOW)

INTENT PosGDI(gen457) Go: YES, active
(Hole1_3,_) -> concentric -> (1): (Hole1_1,_)

INTENT StrGDI(gen458) Go: YES, active
(Hole1_3,_) -> co-radius -> (1): (Hole1_1,_)

INTENT VDI(gen552) Go: YES, intentional
(Hole1_1,_) -> obsoleted_by -> (1): (TSlotTop,RED)

INTENT VDI(gen553) Go: YES, intentional
(Hole1_3,_) -> obsoleted_by -> (1): (TSlotTop,RED)

INTENT HieGDI(gen611) Go: YES, active
(TSlotUnder,PINK) -> nested@side -> (1): (TSlotTop,RED)

INTENT AOI(gen612) Go: YES, active
(TSlotUnder,PINK) -> t_slot -> (1): (TSlotTop,RED)

```

Figure 9-17: Some Recognised *Intentions*.

By analysing Figure 9-17, it can be inferred that the Hole1 feature was split into three features which are subsequently all made *obsoleted_by* another two features (for two different reasons, i.e. interactions). This means that if the features that caused them to become obsolete (USlot-Top and TSlotTop) are subsequently deleted, **FRIEND** will merge them together and create another *hole* feature with the same *volumetric intention* as the former *hole1* feature (although it is considered not to be the original Hole1 feature because of historical reasons).

Some of the FbDI's highlighted in Figure 9-17 (such as Hole1_3 concentric Hole1_1) are kept active because both features have *intentional* status. In this way, if any of the features involved are later reactivated the formerly identified intent would be there and the designer would not be consulted on this again.

```

INTERACTION VI(gen546) Go: NO, active
(TSlotTop,RED) -> general -> (1): (USlot-Top,YELLOW)

INTERACTION VI(gen549) Go: NO, active
(TSlotTop,RED) -> adjoint -> (1): (TSlotUnder,PINK)

INTERACTION VI(gen550) Go: NO, active
(Hole1_3,_) -> limited -> (1): (TSlotTop,RED)

INTERACTION VI(gen551) Go: NO, active
(Hole1_1,_) -> limited -> (1): (TSlotTop,RED)

INTERACTION VI(gen435) Go: NO, active
(Hole1_1,_) -> adjoint -> (1): (USlot-Top,YELLOW)

INTERACTION VI(gen439) Go: NO, active
(Hole1_2,_) -> limited -> (1): (USlot-Top,YELLOW)

INTERACTION VI(gen440) Go: NO, active
(Hole1_2,_) -> adjoint -> (1): (Hole1_1,_)

INTERACTION VI(gen444) Go: NO, active
(Hole1_3,_) -> adjoint -> (1): (USlot-Top,YELLOW)

INTERACTION VI(gen446) Go: NO, active
(Hole1_3,_) -> adjoint -> (1): (Hole1_2,_)

INTERACTION VI(gen350) Go: NO, active
(Step2,RED) -> disjoint -> (1): (Step1,GREEN)

INTERACTION VI(gen410) Go: NO, inactive
(Hole1,_) -> cross -> (1): (USlot-Top,YELLOW)

```

Figure 9-18: Some Feature Interactions in Perng and Chang's Part.

Figure 9-18 shows, among other things, that a *general* feature interaction happened between the USlot-Top and TSlotTop features, which means that none of the interaction cases that can be treated elegantly by **FRIEND** could be identified. In addition, the *cross* interaction between Hole1 and USlot-Top is shown with an *inactive* status because it was already used by a reasoning.

9.9 INFORMATION FOR PROCESS PLANNING

9.9.1 VISUALISATION

Figure 9-19 presents a part file used in Mäntyla et al. (1989) to discuss process planning problems. The figure shows the feature volumes for both the invalid (a) and validated (b) part. Both descriptions (presented in Figure 9-20) produce the same visual appearance (Figure 9-21) when the Boolean operations are applied.

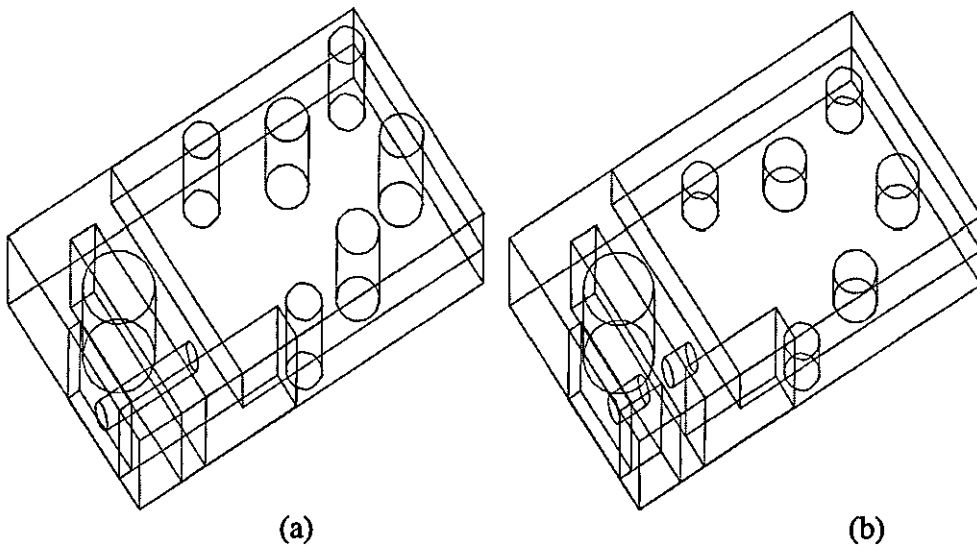


Figure 9-19: Visualisation Without Applying Boolean Operations.

Before and After Validation Reasoning
13 STOCK satellite, rect, 0 orig, orig 0 0 0 57 9 33	(1): STOCK(gen1721),BLUE (satellite, rect, active), BBox: (1:orig,2:orig) P = 0.0 0.0 0.0 P = 57.0 9 0 33 0
Hole1 hole, cyl, 3 orig, orig 38 0 4.5 44 9 10.5	(8): Stp-Up(gen1847),GREEN (step, rect, active), BBox: (1:orig,2:orig) P = 17.0 4.5 0.0 P = 57.0 9.0 33.0
Hole2 hole, cyl, 3 orig, orig 38 0 22.5 44 9 28.5	(10): Hole1_2(gen1894),RED (hole, cyl, active), BBox: (1:orig,2:orig) P = 38.0 0 0 4.5 P = 44.0 4.5 10.5
	(12): Hole2_2(gen1925),YELLOW (hole, cyl, active), BBox: (1:orig,2:orig) P = 38 0 0.0 22.5 P = 44 0 4.5 28 5

Hole3 hole, cyl, 3 orig, orig 50 0 12 56 9 18	(14): Hole3_2(gen1956),PINK (hole, cyl, active), BBox: (1:orig,2:orig) P = 50 0 0 0 12.0 P = 56 0 4.5 18.0
Holea hole, cyl, 2.5 orig, orig 27 0 1 32 9 6	(16): Holea_2(gen1987),RED (hole, cyl, active), BBox: (1:orig,2:orig) P = 27.0 0 0 1.0 P = 32.0 4.5 6.0
Holeb hole, cyl, 2.5 orig, orig 27 0 27 32 9 32	(18): Holeb_2(gen2018),LIGHTBLUE (hole, cyl, active), BBox: (1:orig,2:orig) P = 27.0 0 0 27.0 P = 32.0 4.5 32 0
Holec hole, cyl, 2.5 orig, orig 51 0 27 56 9 32	(20): Holec_2(gen2049),BLACK (hole, cyl, active), BBox: (1:orig,2:orig) P = 51.0 0 0 27 0 P = 56.0 4.5 32.0
Stp-Up step, rect, 0 orig, rx90 17 4 5 0 57 9 33	(22): Slot(gen2152),GRAY (slot_tru, rect, active), BBox: (1:orig,2:rx270) P = 7 0 0 0 0 0 P = 11.0 9.0 28.0
HoleTransversal hole, cyl, 1.5 rx270, ry90 0 3 8 15 6 11	(24). Stp-Comer_1(gen2235),RED (face, rect, active), BBox: (1:orig,2:rx270) P = 0 0 0 0 0 0 P = 7.0 9.0 5.0
Slot slot, rect, 0 orig, orig 7 0 0 11 9 28	(26): Stp-Comer_3(gen2247),DARKGREEN (step, rect, active), BBox: (1:rx90,2:ry180) P = 11.0 0.0 0 0 P = 26.0 9 0 5.0
Stp-Comer slot, rect, 0 orig, orig 0 0 0 26 9 5	(27): Step-front(gen2320),GREEN (step, rect, active), BBox: (1:rx90,2:ry180) P = 0 0 0 0 5.0 P = 2 0 9.0 18 0
Step-front step, rect, 0 orig, rz270 0 0 5 2 9 18	(30): HoleTransversal_2_1(gen2420),BLACK (hole, cyl, active), BBox: (1:orig,2:rz90) P = 2 0 3.0 8 0 P = 7 0 6 0 11.0
Holao hole, cyl, 5 orig, ry270 4 0 14 14 9 24	(32): HoleTransversal_2_3(gen2432),GRAY (pocket, cyl, active), BBox: (1:orig,2:rz90) P = 11.0 3.0 8.0 P = 15.0 6.0 11.0
	(33): Holao(gen2513),LIGHTBLUE (hole, cyl, active), BBox: (1:orig,2:ry270) P = 4 0 0.0 14.0 P = 14.0 9.0 24.0

Figure 9-20: Mäntylä et al's Part Descriptions.

Some RDI's can be obtained from this part and can be used for process planning. In particular, the *co-radius*, *parallel* and *concentric* GDI's could

help identify groups of *hole* features to be machined in the same setup and even with the same process and tool.

The last *co-radius* GDI and the concentric GDI (both presented in bold in Figure 9-22) were obtained through guided *enrichment* (because they were both originally split from the same feature, see section 8.10.1). All the other *co-radius* GDI's were obtained from blind *enrichment* rules (see section 8.10.2).

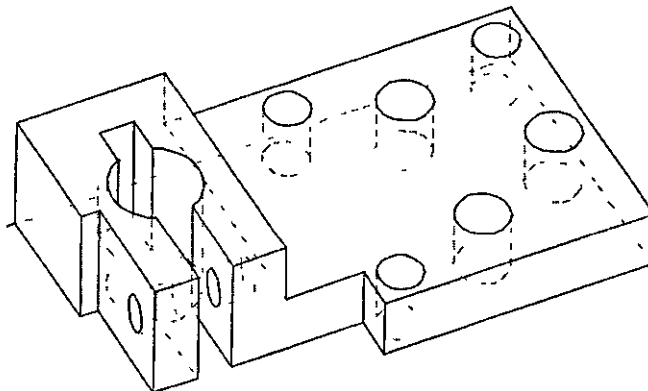


Figure 9-21: Mäntylä' et al's Part.

FRIEND also recognised a possible accessibility problem (see section 8.9.4 and Figure 8-22) when splitting the “HoleTransversal” feature. However, it was considered acceptable just to be able to exemplify in this same part the various *enrichment* reasonings.

```

(Holec_2,BLACK) -> nested@bot -> (1): (Stp-Up,GREEN)
(Holec_2,BLACK) -> parallel -> (1): (Holeb_2,LIGHTBLUE)
(Holeb_2,LIGHTBLUE) -> nested@bot -> (1): (Stp-Up,GREEN)
(Holea_2,RED) -> nested@bot -> (1): (Stp-Up,GREEN)
(Hole3_2,PINK) -> nested@bot -> (1): (Stp-Up,GREEN)
(Hole2_2,YELLOW) -> nested@bot -> (1): (Stp-Up,GREEN)
(Hole1_2,RED) -> nested@bot -> (1): (Stp-Up,GREEN)
(Holec_2,BLACK) -> co-radius -> (1): (Holeb_2,LIGHTBLUE)
(Holec_2,BLACK) -> co-radius -> (1): (Holea_2,RED)
(Holeb_2,LIGHTBLUE) -> co-radius -> (1): (Holea_2,RED)
(Hole3_2,PINK) -> co-radius -> (1): (Hole2_2,YELLOW)
(Hole3_2,PINK) -> co-radius -> (1): (Hole1_2,RED)
(Hole2_2,YELLOW) -> co-radius -> (1): (Hole1_2,RED)
(Holec_2,BLACK) -> parallel -> (1): (Holea_2,RED)
(Holec_2,BLACK) -> parallel -> (1): (Hole3_2,PINK)
(Holec_2,BLACK) -> parallel -> (1): (Hole2_2,YELLOW)
(Holec_2,BLACK) -> parallel -> (1): (Hole1_2,RED)
(Holeb_2,LIGHTBLUE) -> parallel -> (1): (Holea_2,RED)
(Holeb_2,LIGHTBLUE) -> parallel -> (1): (Hole3_2,PINK)
(Holeb_2,LIGHTBLUE) -> parallel -> (1): (Hole2_2,YELLOW)
(Holeb_2,LIGHTBLUE) -> parallel -> (1): (Hole1_2,RED)
(Holea_2,RED) -> parallel -> (1): (Hole3_2,PINK)
(Holea_2,RED) -> parallel -> (1): (Hole2_2,YELLOW)
(Holea_2,RED) -> parallel -> (1): (Hole1_2,RED)
(Hole3_2,PINK) -> parallel -> (1): (Hole2_2,YELLOW)
(Hole3_2,PINK) -> parallel -> (1): (Hole1_2,RED)
(Hole2_2,YELLOW) -> parallel -> (1): (Hole1_2,RED)
(Stp-Corner_1,RED) -> nested@side -> (1): (Slot,GRAY)
(Stp-Corner_3,DARKGREEN) -> nested@side -> (1): (Slot,GRAY)
(Stp-Corner_1,RED) -> parallel -> (1): (Stp-Up,GREEN)
(HoleTransversal_2_1,BLACK) -> nested@side -> (1): (Slot,GRAY)
(HoleTransversal_2_1,BLACK) -> nested@bot -> (1): (Step-front,GREEN)
(HoleTransversal_2_3,GRAY) -> nested@side -> (1): (Slot,GRAY)
(HoleTransversal_2_3,GRAY) -> parallel -> (1): (HoleTransversal_2_1,BLACK)
(HoleTransversal_2_3,GRAY) -> concentric -> (1): (HoleTransversal_2_1,BLACK)
(Step-front,GREEN) -> nested@bot -> (1): (Stp-Corner_1,RED)
(HoleTransversal_2_3,GRAY) -> co-radius -> (1): (HoleTransversal_2_1,BLACK)

```

Figure 9-22: RDI's for Mäntylä et al's Part.

9.9.2 REDESIGN

Figure 9-23 presents an example of a slotted cross-shaped feature-based part built and validated by **FRIEND**. This part was adapted from a study reported in Das96 on setup planning and automated redesign suggestions. Figure 9-24 presents a list of all valid features output by **FRIEND** after the validation reasoning.

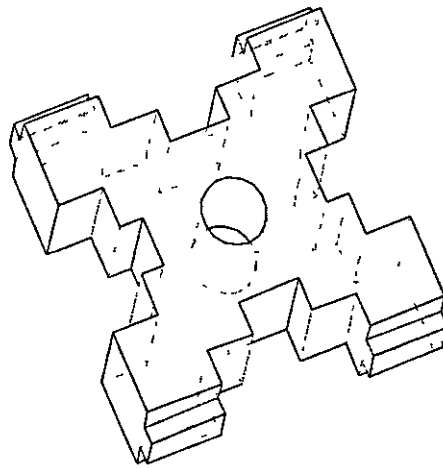


Figure 9-23: A Slotted Cross-Shaped Part.

Note that all features have a quadrangular volume type, except the *hole* feature. A redesign suggestion can be made by considering the FbDI's in Figure 9-25 to obtain various alternative representations including some cylindrical volume types. This simple change can produce a better part from the cost and time savings perspective.

(1): STOCK(gen1),BLUE (satellite, rect, active), BBox: (1:orig,2:orig)
P = 0 0 0 0 0 P = 20 0 5.0 20.0

(6): CentreHole(gen177),GREEN (hole, cyl, active), BBox: (1:orig,2:orig)
P = 8 0 0 0 8 0 P = 12.0 5.0 12.0

(7): SL1(gen190),RED (slot_tru, rect, active), BBox: (1:orig,2:rx270)
P = 5.0 0.0 0 0 P = 15 0 5.0 3.0

(8): STP1_1(gen207),YELLOW (step, rect, active), BBox: (1:rx180,2:ry90)
P = 0.0 0.0 0.0 P = 5.0 1.5 1.0

(10): STP1_3(gen219),PINK (step, rect, active), BBox: (1:rx180,2:ry90)
P = 15 0 0 0 0 P = 20 0 1.5 1.0

(11): STP2_1(gen256),RED (step, rect, active), BBox: (1:orig,2:ry90)
P = 0.0 3.5 0 0 P = 5 0 5.0 1.0

(13): STP2_3(gen268),LIGHTBLUE (step, rect, active), BBox: (1:orig,2:ry90)
P = 15 0 3 5 0 0 P = 20 0 5 0 1.0

(14): SL2(gen308),BLACK (slot_tru, rect, active), BBox: (1:orig,2:rx90)
P = 8.0 0.0 15.0 P = 12.0 5.0 17.0

(15): SL3(gen361),GRAY (slot_tru, rect, active), BBox: (1:orig,2:rx90)
P = 5.0 0.0 17 0 P = 15.0 5.0 20.0

(16): STP4_1(gen413),RED (step, rect, active), BBox: (1:orig,2:ry270)
P = 0 0 3 5 19.0 P = 5 0 5 0 20.0

(18): STP4_3(gen425),DARKGREEN (step, rect, active), BBox: (1:orig,2:ry270)
P = 15 0 3 5 19.0 P = 20.0 5.0 20.0

(19): STP3_1(gen474),GREEN (step, rect, active), BBox: (1:rx90,2:rz270)
P = 0 0 0 0 19 0 P = 5 0 1.5 20.0

(21): STP3_3(gen486),BLACK (step, rect, active), BBox: (1:rx90,2:rz270)
P = 15.0 0 0 19 0 P = 20.0 1.5 20 0

(22): SL4(gen602),GRAY (slot_tru, rect, active), BBox: (1:orig,2:rx270)
P = 8.0 0.0 3 0 P = 12.0 5.0 5 0

(24): SLlateral1(gen675),LIGHTBLUE (slot_tru, rect, active), BBox: (1:rx90,2:ry270)
P = 0.0 0.0 6 0 P = 3 0 5.0 14 0

(25): SLlateral2(gen746),LIGHTGRAY (slot_tru, rect, active), BBox: (1:rx90,2:ry270)
P = 3 0 0 0 8.5 P = 4.5 5.0 11.5

(26): SLlateral3(gen831),BLUE (slot_tru, rect, active), BBox: (1:rx90,2:ry90)
P = 17.0 0.0 6.0 P = 20.0 5.0 14.0

(27): SLlateral4(gen898),GREEN (slot_tru, rect, active), BBox: (1:rx90,2:ry90)
P = 15.5 0.0 8.5 P = 17.0 5.0 11.5

Figure 9-24: Validated Output for the Cross-Shaped Part.


```

(STP3_1, GREEN) -> nested@side -> (1): (SL3, GRAY)
(STP3_3, BLACK) -> nested@side -> (1): (SL3, GRAY)
(STP4_1, RED) -> nested@side -> (1): (SL3, GRAY)
(STP4_3, DARKGREEN) -> nested@side -> (1): (SL3, GRAY)
(STP2_1, RED) -> nested@side -> (1): (SL1, RED)
(STP2_3, LIGHTBLUE) -> nested@side -> (1): (SL1, RED)
(STP1_1, YELLOW) -> nested@side -> (1): (SL1, RED)
(STP1_3, PINK) -> nested@side -> (1): (SL1, RED)

(SL2, BLACK) -> nested@bot -> (1): (SL3, GRAY)
(SL4, GRAY) -> nested@bot -> (1): (SL1, RED)
(SLlateral2, LIGHTGRAY) -> nested@bot -> (1): (SLlateral1, LIGHTBLUE)
(SLlateral4, GREEN) -> nested@bot -> (1): (SLlateral3, BLUE)

(STP3_3, BLACK) -> coEAD -> (1): (STP3_1, GREEN)
(STP3_3, BLACK) -> coEAD -> (1): (STP4_3, DARKGREEN)
(STP3_3, BLACK) -> coEAD -> (1): (STP4_1, RED)
(STP3_3, BLACK) -> coEAD -> (1): (STP1_3, PINK)
(STP3_3, BLACK) -> coEAD -> (1): (STP1_1, YELLOW)
(STP3_1, GREEN) -> coEAD -> (1): (STP4_3, DARKGREEN)
(STP3_1, GREEN) -> coEAD -> (1): (STP4_1, RED)
(STP3_1, GREEN) -> coEAD -> (1): (STP1_3, PINK)
(STP3_1, GREEN) -> coEAD -> (1): (STP1_1, YELLOW)
(STP4_3, DARKGREEN) -> coEAD -> (1): (STP4_1, RED)
(STP4_3, DARKGREEN) -> coEAD -> (1): (STP2_3, LIGHTBLUE)
(STP4_3, DARKGREEN) -> coEAD -> (1): (STP2_1, RED)
(STP4_1, RED) -> coEAD -> (1): (STP2_3, LIGHTBLUE)
(STP4_1, RED) -> coEAD -> (1): (STP2_1, RED)
(SL3, GRAY) -> coEAD -> (1): (SL2, BLACK)
(STP2_3, LIGHTBLUE) -> coEAD -> (1): (STP2_1, RED)
(STP2_3, LIGHTBLUE) -> coEAD -> (1): (STP1_3, PINK)
(STP2_3, LIGHTBLUE) -> coEAD -> (1): (STP1_1, YELLOW)
(STP2_1, RED) -> coEAD -> (1): (STP1_3, PINK)
(STP2_1, RED) -> coEAD -> (1): (STP1_1, YELLOW)
(STP1_3, PINK) -> coEAD -> (1): (STP1_1, YELLOW)
(SL4, GRAY) -> coEAD -> (1): (SL1, RED)
(SLlateral2, LIGHTGRAY) -> coEAD -> (1): (SLlateral1, LIGHTBLUE)
(SLlateral4, GREEN) -> coEAD -> (1): (SLlateral3, BLUE)

```

Figure 9-25: Some FbDI's for the Cross-Shaped Part.

9.10 EDINBURGH COMPOSITE COMPONENT

Figure 9-26 presents the Edinburgh Composite Component found in Mill93 and defined as a test part for process planning conflict situations. Again, although **FRIEND** does not generate a process plan it obtains a plethora of information that can help analyse and solve some of the planning difficulties. This valuable extra information comes in the form of VDI's (Figure 9-28), GDI's (Figure 9-29) and AOI's (Figure 9-30).

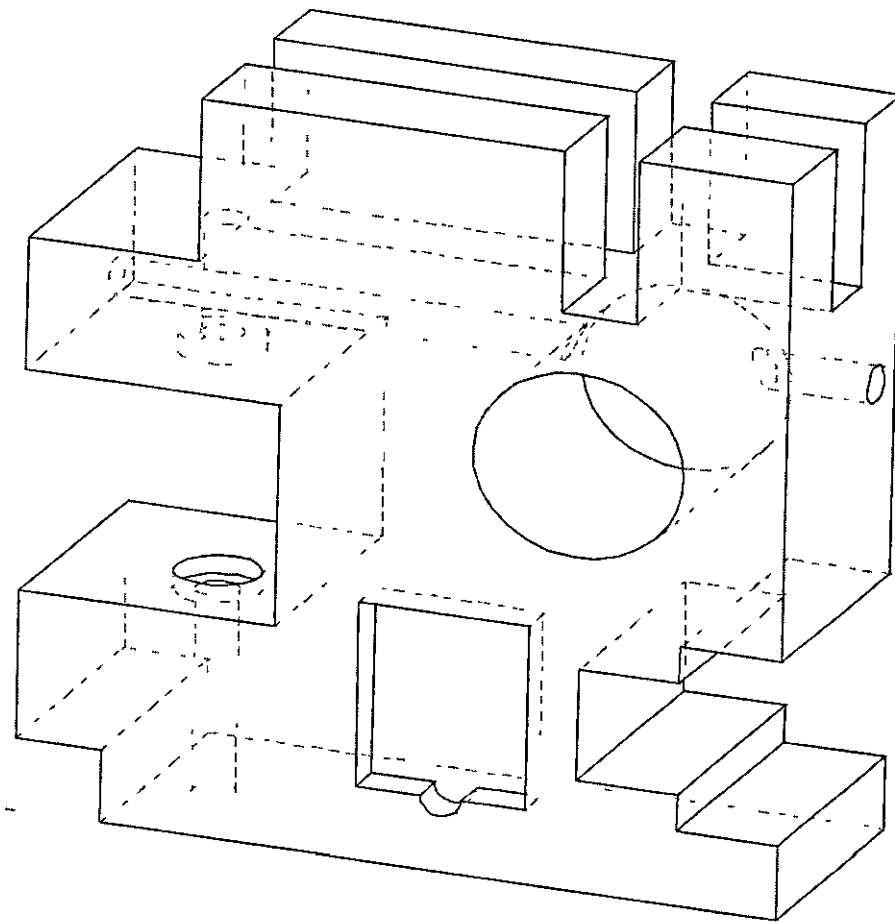


Figure 9-26: The Edinburgh Composite Component.

```

(1): STOCK(gen718),BLUE (satellite, rect, active), BBox: (1:orig,2:orig)
P = 0 0 0 0 0 0 P = 33 0 100 0 90.0

(2): CutOutBig(gen724),GREEN (pocket, rect, active), BBox: (1:orig,2:rz90)
P = 0 0 10.0 30 0 P = 5.0 35 0 50.0

(3): CutOutHole(gen732),RED (pocket, cyl, active), BBox: (1:orig,2:rz90)
P = 0.0 7.0 37.0 P = 5.0 13.0 43.0

(4): TopCrossA(gen763),YELLOW (slot_tru, rect, active), BBox: (1:orig,2:orig)
P = 13.0 78.0 0.0 P = 22.0 100.0 70 0

(5): TopCrossB(gen774),PINK (slot_tru, rect, active), BBox: (1:orig,2:ry90)
P = 0 0 78.0 18 0 P = 33.0 100 0 27.0

(6): BottomCross(gen802),RED (step, rect, active), BBox: (1:orig,2:ry270)
P = 0.0 78.0 70 0 P = 33.0 100 0 90.0

(7): BigMiddleSlot(gen835),LIGHTBLUE (slot_tru, rect, active), BBox: (1:rx90,2:rz90)
P = 0.0 30.0 60 0 P = 33.0 60 0 90 0

(8): BigHole(gen866),BLACK (hole, cyl, active), BBox: (1:orig,2:rz90)
P = 0.0 45.0 12.0 P = 33.0 70.0 37.0

(9): CrossingHole(gen887),GRAY (hole, cyl, active), BBox: (1:rx90,2:orig)
P = 25 0 60 0 0.0 P = 30.0 65.0 90.0

(10): NestedSlotBig(gen905),RED (slot_tru, rect, active), BBox: (1:rx270,2:rz90)
P = 0.0 10.0 0.0 P = 33.0 35.0 12.0

(11): NestedSlotSmall(gen959),DARKGREEN (slot_tru, rect, active), BBox: (1:rx270,2:rz90)
P = 0.0 15.0 12.0 P = 33.0 30.0 24.0

(12): CBoreHole(gen1029),_ (hollow, cyl, inactive), BBox: (1:orig,2:orig)
P = 10.0 27.5 67.5 P = 20.0 62.5 77.5

(13): CBoreHole_1(gen1059),GREEN (pocket, cyl, active), BBox: (1:orig,2:orig)
P = 10 0 27 5 67.5 P = 20 0 30 0 77.5

(15): CBoreHole_3(gen1071),BLACK (pocket, cyl, active), BBox: (1:orig,2:rx180)
P = 10.0 60.0 67.5 P = 20.0 62.5 77.5

(21): ThinContiguityHole_3_2(gen1232),GRAY (hole, cyl, active), BBox: (1:orig,2:orig)
P = 12.5 62.5 70.0 P = 17.5 78 0 75.0

(23): ThinContiguityHole_1_2(gen1285),LIGHTBLUE (hole, cyl, active), BBox: (1:orig,2:orig)
P = 12.5 0.0 70.0 P = 17.5 27.5 75 0

(24): BottomStep(gen1350),LIGHTGRAY (step, rect, active), BBox: (1:rx90,2:rz270)
P = 0.0 0.0 80.0 P = 33 0 10 0 90.0

```

Figure 9-27: The Output for the Edinburgh Composite Component.

```

(CBoreHole,_) -> split_into -> (2): (CBoreHole_1, GREEN) (CBoreHole_2,_)
(CBoreHole_3, BLACK)

(ThinContiguityHole,_) -> split_into -> (2): (ThinContiguityHole_1,_)
(ThinContiguityHole_2,_) (ThinContiguityHole_3,_)

(ThinContiguityHole_3,_) -> split_into -> (2): (ThinContiguityHole_3_1,_)
(ThinContiguityHole_3_2, GRAY)

(ThinContiguityHole_1,_) -> split_into -> (2): (ThinContiguityHole_1_1,_)
(ThinContiguityHole_1_2, LIGHTBLUE)

```

Figure 9-28: VDI's Gathered from the Edinburgh Composite Component.

```

(TopCrossA, YELLOW) -> nested@bot -> (1): (BottomCross, RED)
(NestedSlotSmall, DARKGREEN) -> nested@bot -> (1): (NestedSlotBig, RED)
(NestedSlotBig, RED) -> parallel -> (1): (BottomCross, RED)
(NestedSlotBig, RED) -> parallel -> (1): (CutOutBig, GREEN)
(CBoreHole_3, BLACK) -> co-radius -> (1): (CBoreHole_1, GREEN)
(CBoreHole_1, GREEN) -> nested@side -> (1): (BigMiddleSlot, LIGHTBLUE)
(CBoreHole_3, BLACK) -> nested@side -> (1): (BigMiddleSlot, LIGHTBLUE)
(CBoreHole_3, BLACK) -> parallel -> (1): (CBoreHole_1, GREEN)
(CBoreHole_3, BLACK) -> concentric -> (1): (CBoreHole_1, GREEN)
(ThinContiguityHole_1_2, LIGHTBLUE) -> nested@bot -> (1): (CBoreHole_1, GREEN)
(ThinContiguityHole_1_2, LIGHTBLUE) -> parallel -> (1): (ThinContiguityHole_3_2, GRAY)
(ThinContiguityHole_1_2, LIGHTBLUE) -> concentric -> (1): (ThinContiguityHole_3_2, GRAY)
(ThinContiguityHole_3_2, GRAY) -> nested@bot -> (1): (BottomCross, RED)
(ThinContiguityHole_3_2, GRAY) -> nested@bot -> (1): (CBoreHole_3, BLACK)
(ThinContiguityHole_1_2, LIGHTBLUE) -> co-radius -> (1): (ThinContiguityHole_3_2, GRAY)
(ThinContiguityHole_1_2, LIGHTBLUE) -> co-radius -> (1): (CrossingHole, GRAY)
(ThinContiguityHole_3_2, GRAY) -> co-radius -> (1): (CrossingHole, GRAY)
(ThinContiguityHole_1_2, LIGHTBLUE) -> parallel -> (1): (CBoreHole_1, GREEN)
(ThinContiguityHole_1_2, LIGHTBLUE) -> concentric -> (1): (CBoreHole_1, GREEN)
(ThinContiguityHole_1_2, LIGHTBLUE) -> parallel -> (1): (CBoreHole_3, BLACK)
(ThinContiguityHole_1_2, LIGHTBLUE) -> concentric -> (1): (CBoreHole_3, BLACK)
(ThinContiguityHole_3_2, GRAY) -> parallel -> (1): (CBoreHole_1, GREEN)
(ThinContiguityHole_3_2, GRAY) -> concentric -> (1): (CBoreHole_1, GREEN)
(ThinContiguityHole_3_2, GRAY) -> parallel -> (1): (CBoreHole_3, BLACK)
(ThinContiguityHole_3_2, GRAY) -> concentric -> (1): (CBoreHole_3, BLACK)
(BottomStep, LIGHTGRAY) -> parallel -> (1): (TopCrossB, PINK)
(BottomStep, LIGHTGRAY) -> parallel -> (1): (CutOutHole, RED)
(BottomStep, LIGHTGRAY) -> parallel -> (1): (CutOutBig, GREEN)

```

Figure 9-29: GDI's Gathered from the Edinburgh Test Component.

```

INTENT AOI(gen759) Go: YES, active
(CutOutHole,RED) -> cut_out -> (1): (CutOutBig,GREEN)

INTENT AOI(gen760) Go: YES, active
(CutOutHole,RED) -> e_feat -> (1): (CutOutBig,GREEN)

INTENT AOI(gen761) Go: YES, active
(CutOutHole,RED) -> coEAD -> (1): (CutOutBig,GREEN)

INTENT AOI(gen799) Go: YES, active
(TopCrossB,PINK) -> x_feat -> (1): (TopCrossA,YELLOW)

INTENT AOI(gen830) Go: YES, active
(TopCrossA,YELLOW) -> coEAD -> (1): (TopCrossB,PINK)

INTENT AOI(gen903) Go: YES, active
(CrossingHole,GRAY) -> x_feat -> (1): (BigHole,BLACK)

INTENT AOI(gen1014) Go: YES, active
(NestedSlotSmall,DARKGREEN) -> coEAD -> (1): (NestedSlotBig,RED)

INTENT AOI(gen1332) Go: YES, active
(ThinContiguityHole_1_2,LIGHTBLUE) -> coEAD -> (1): (ThinContiguityHole_3_2,GRAY)

INTENT AOI(gen1335) Go: YES, active
(ThinContiguityHole_1_2,LIGHTBLUE) -> c_bore -> (1): (ThinContiguityHole_3_2,GRAY)

INTENT AOI(gen1347) Go: YES, active
(ThinContiguityHole_3_2,GRAY) -> c_bore -> (1): (CBoreHole_1,GREEN)

INTENT AOI(gen1411) Go: YES, active
(BottomStep,LIGHTGRAY) -> coEAD -> (1): (BottomCross,RED)

```

Figure 9-30: AOI's Gathered from the Edinburgh Composite Component.

The major concern of **FRIEND** is to make explicit to the designer some of the intentions (the FbDI's) that can be assigned to the model such as *nested@side*, *parallelism*, *x_feat*, *cut-out* and *c_bore* (see Figure 9-29 and Figure 9-30). No strategy for planning the processing or production of the part is suggested.

9.11 A COMPARISON PROBLEM

It was found that a comparison between the functionalities of **FRIEND** and other systems is not straightforward because most of the systems studied perform some sort of geometric reasoning on the complete model (and therefore, as a post-processing procedure) while **FRIEND** accumulates knowledge throughout the design process because it analyses the part model every time an operation is performed.

Furthermore, some of the test cases presented were obtained from literature more interested in feature-based process planning problems (of the complete part model) while the major concern in **FRIEND** is in the correctness of the representation and the FbDI's that can be gathered from and during the design process.

In doing this, **FRIEND** is capable of producing much more information than most feature-based modellers and this information can be used for various engineering-related activities, not only process planning. Some parts of this reasoning are straight derivations from the feature-based designer's intents (FbDI's) identified by **FRIEND** and others would require extra technological information to reach their conclusion.

9.12 SUMMARY

This chapter presented some test parts that were adapted from parts found in the literature. **FRIEND** could model them and although the production of process plans was not the objective, it was able to correct some of the definition mistakes (introduced deliberately) and to produce a plethora of information that could help downstream applications such as process planning.

Some difficulties were found in comparing the functionality of **FRIEND** with other work because **FRIEND** gathers intentions during the ongoing feature-based modelling task while most of the other systems perform a post-processing analysis on the final and static feature-based model.

It can be inferred that the way the model is built can affect the resulting amount and type of information produced by **FRIEND** and this is consistent with the non-commutability characteristics of the Boolean operations (which are implied by the feature-based models).

10. DISCUSSION

This chapter critically discusses some of the main aspects and findings regarding the concepts presented, the implementation carried out and the results from the test cases.

10.1 THE VALIDATION FRAMEWORK

The validation framework (Figure 3-2) is simple but emphasises the fact that the *domain characterisation (FbDI's)* influences the *validity conditions* (reasoning), often helped by the feature interaction identification methodology and the validity of the model, which subsequently influences the definition of adequate (revalidation) *operations*.

The significance of this interconnection is that it suggests how the domain might be further extended to include other behaviours. For instance, if a new FbDI is to be considered, it must be ensured that the appropriate reasoning can be produced to test the suitability of existing feature interaction cases and analysis operations. As new invalidity tests are likely to emerge, this also tests the suitability of the existing set of revalidation operations.

The occurrence of any unsuitability raises the question as to why it has happened, what actions need to be taken and how does the new FbDI fit into the classification? If it does not fit, a reason has to be sought and methods for adapting the classification to accommodate the newly added element should also be addressed.

This is a similar scenario to that encountered when a new feature type or parameter is added which leads to a consideration of the suitability of modelling and editing operations.

The validation framework, and its implementation in **FRIEND**, implies a Hybrid DbF and FeR (**HDR**) approach to some extent and it has been made clear that a thorough implementation of the ideas presented here would require a FeR subsystem.

It was also realised that the validity depends on local analysis of the ongoing design as well as global analysis of the finished model (following the lines of work presented in Requicha89b and Rossignac90).

10.2 INTENT-DRIVEN VERSUS CONSTRAINT-DRIVEN ONLY APPROACHES

Approaching a design using a geometric constraining system only represents a more rigid approach compared to an intent-driven one because it demands a considerable effort in understanding the concepts, equations and relations involved in a particular design and to establish all fixed constraints (with the exception of Mäntylä94) and it can be difficult to edit (numeric, geometric or algebraic) constraints. In addition, the resulting information (constraint graphs) may be hard-coded for the particular design, and thus its reuse in another similar design is difficult.

The use of a geometric constraining approach has already been introduced into FBM (Nielsen91, Sheu93, Mäntylä94, Dohmen94, Shah94b) to either represent features as basic relational elements or to establish relationships between geometrical constituent elements of different features. However, the approach is still based on conventional parametric or variational constraining methods

using mainly low-level geometric entities such as points, edges and faces. Thus, PDI's have not been detailed in the current FbDI taxonomy.

Parametric FbDI's (geometric constraints) are concerned with any feature, "sub-feature" (faces, edges or vertices) and feature parameters on the same part or between different parts. In contrast, other types of FbDI's are specified between features themselves through some well-defined high-level relationships, which together with an experience-based "non-blind" search, can lead to a "intent-recognition" procedure enriching the representation and alleviating the designer of such tedious tasks.

FbDI's can be used to reason about the design knowledge and structure and are not restricted to the derivation of parameter or dimension values. FbDI's are thus considered a generalisation of constraints where not only fixed algebraic and geometric relationships are considered but also other engineering-related relations (such as *nested@side* GDI and *x_feat* AOI) are included.

If a FbDI becomes *active*, it can be considered to have become a "loose constraint" where its main purpose is to make explicit to the designer what could have been forgotten in terms of intentions to be achieved. This means that an intent-driven environment is more forgiving when changes to existing FbDI's are required. The system accepts changes to the existing configuration of intents regardless of how drastic they are, can ask for confirmation, could highlight the effects of the manipulation and even suggest further actions.

The use of a geometric constraint mechanism could enhance the functionality of **FRIEND** because it could explicitly capture designer's intent in the form of parameter or dimension relationships. However, due to the many intermediate stages that a design needs to undergo and the trial-and-error nature of design practices, FbDI's in general have a more dynamic behaviour, being created and deleted according to the stage of the design.

Thereupon, it can be said that a geometric constraint-driven approach is more suitable for a performance and function-oriented intricate design while the intent-driven approach (without PDI's) would suit a more application and history-oriented design. It is believed that the intent-driven and constraint-driven approaches can work alongside and indeed, be complementary to each other.

10.3 USE OF THE ELICITATION PROCESS

The entity *elicitation process* that was applied to features (section 1.5) represents a loose formalism that helped in obtaining a more detailed description of the validation framework elements. Although most of the individual components were gathered from the literature, they fit well in the classifications and consequent taxonomies.

The application of the elicitation process, as defined here for featurization, to the items of information required for the validation task needed no adjustments. Both *validation* and *elicitation criteria* (Figure 1-8) were, however, difficult to devise for all cases. Even for features themselves there is no agreement on what such criteria might be. Possibly a more practical implementation with its consequently more tangible limitations and objectives could help produce a more tangible set of criteria.

10.4 DESIGNER'S INTENTS

A FBM system driven by FbDI's is considered of high significance because it could help to preserve the reasons for a particular decision in a design. For instance, the reason for a feature to be located at a specific position could be an *axial symmetry* structural GDI.

It could be beneficial to future FBM systems if “featurization” analysis (section 1.5.2) and feature taxonomies are accompanied by similar “intenturization” analyses (section 4.2) and FbDI’s taxonomies to help establish more meaningful and precise implementation boundaries and the capabilities of a particular feature-based application.

Feature-based Designer’s Intents (FbDI’s) were divided into three areas: related to individual features (the VDI’s); related to groups of features (the geometrical RDI’s and PDI’s), and; dependent on applications (the application-oriented RDI’s).

Validation reasoning was carried out at three levels: at the feature class level, which is defined for all features; at the feature object level, which is defined for different feature types or instances, and at the application level that, not only considers the two previous levels, but also considers application restraints.

Some degree of automation can be assigned to the validation process. Most of the feature validation could be automated at the feature class level but there are some reasonings at the feature type level that are cumbersome to automate. However, the assistance of the designer is necessary because not all of his intentions can be captured or, most importantly, predicted (see Zhang93).

It can be inferred from the analysis of Figure 7-6 that some of the validation could indeed be defined within the feature class itself but as the analysis goes further down, more and more of the analysis starts to consider feature type information (e.g. is it a *hole* or a *pocket*) and feature type specific information (making distinctions between the centre radius of a *hole* and the fillet radius of a *slot* feature). Also, as the analysis goes from top to bottom in Figure 7-6, the focus is changed from a more individual level of understanding towards a more

collective perspective at the same time as it goes from feature dependent towards application dependent reasoning.

Automatic recognition of pre-established FbDI's and the consequent representation *enrichment* can be achieved, raising the quality and usefulness of the model as well as relieving the designer of these tedious tasks. Feature-based intent recognition, or representation *enrichment*, is a powerful resource that would facilitate "intelligent" reasoning.

10.5 FEATURE INTERACTIONS

The feature interaction classification and identification methodology presented here has several advantages:

- It is a DbF-aware scheme and encompasses existing approaches and classifications (both from FeR systems as well as from narrow DbF domains), and it adds a comprehensive unbound (and therefore, unbiased) coverage and a clarification of the *interference* and *interaction* terms .
- It is multi-level, which allows reasoning to be performed at three levels. The cases are as detailed and accurate as required allowing specific actions to be taken for apparently similar cases and also allowing it to be used by, and separated from, different application analyses. All levels share the same structure and concept of classification (except for a few minor details, section 5.4.6) promoting the coherence of the scheme. This also avoids misunderstandings because there is no mixing of geometric entities of different dimensionality at each level.
- Particular attention was paid to choosing words to describe each category with the aim of producing a clearer, more meaningful and easier to understand vocabulary. The categories have been formally defined through simple rules using commonly available Boolean operators and tests, which

facilitates integration with GSM modellers. Thus, categories are defined separately and independent of the underlying GSM implementation.

- Neither concave/convex nor planar/non-planar assumptions are made as in the case in much research work. This is to the minor detriment of efficiency, but many of the operations and tests can be quickly and accurately predicted using bounding boxes.

The feature interaction methodology allows a FBM system to build a rich scenario of the model including information about the feature's surroundings, and this information can be valuable and beneficial to other applications. Some functional meanings (or *functional* features) can be identified such as when a *slot* is *near VI* to the stock's end it results in a *wall* feature or when a *slot* is *near VI* to another *slot* it results in a *rib* feature. The totally different functional purposes of *wall* and *rib* features (Lee94) can be inferred from the model and subsequently considered in further applications/analysis.

10.6 FEATURE OPERATIONS

Feature-based operations have received little attention in the literature possibly because it has been considered more an implementation issue than a research one. The feature operations taxonomy presented here emphasises the fact that much has to be done to facilitate the manipulation (editing, modelling and revalidation) of feature-based models.

The variety of feature operations presented in the taxonomy makes the two usually available operations, add and delete feature, seem so limited and limiting. In reality, even considering only these two feature operations, it can be inferred that there are many other types of operations that need to be specified and implemented that could render the system implementation not trivial at all.

Being aware of this variety is an important aspect of a design-by-features system implementation in order to be able to estimate the effort in producing a system that is easy to manipulate.

Moreover, the taxonomy of operations was observed to be dependent on a number of factors:

- Feature interaction cases.

The revalidation operations reflect partially how features are defined (dependent on the *entity set* (Σ^f) used to identify them) and how they are allowed to interact amongst themselves. For instance, it was observed that the *split* revalidation operation could be deployed onto a feature's *volume* automatically if a *subjoint* VI case occurs. On the other hand, an *inside* FI case could use a change face code *to_V* labelling revalidation operation onto a feature face.

- Application domain.

Different characterisations of the domain influence the size of the revalidation operation set. A wider domain possibly needs an extended set of operations. For instance, rotational features would probably frequently use a different set of operations than prismatic features.

- Editing capabilities .

The variety and level of manipulations available to the designer greatly influences the set of revalidation operations to be devised. However, it was found that there is a minimum set of modelling manipulations consisting of *add* and *delete feature* and *add* and *delete FbDI* which can be used internally to implement other editing manipulations.

To provide the full editing flexibility compatible with conventional and GSM CAD systems a DbF system would need a full implementation of a (localised

and/or incremental) FeR system to recognise all possible manipulation results originating from the editing operations. However, the designer might be tempted to abuse the use of the editing operation, which could produce unrecognisable features.

Monitoring the designer's editing capabilities for each case/feature (as proposed before, Case93c) would be a major task, thus systems have instead favoured much simpler editing operations in order to avoid this complexity factor. This emphasises how closely feature operations are related to feature model validation.

A (localised, incremental or somehow optimised) FeR functionality is nevertheless required in the validation approach and the level of optimisation and complexity of the FeR function is dictated by the freedom of manipulation allowed by the DbF function. Validation has been interpreted as, and closely related to, Feature Recognition (Dixon90, Pratt93, Martino94a).

If low-level editing manipulations are considered, such as *chamfering an edge* or *tapering a face*, this greatly adds to the complexity of the revalidation operations. In this case, a taxonomy of operations would include very basic manipulations such as Euler and Sweeping Operators (Stroud93, Subrahmanyam95). Therefore, editing manipulations can add to the flexibility of the system but also certainly add to its complexity.

It was initially thought that editing operations had not been fully implemented elsewhere solely because of implementation issues. However, it was discovered that many operations have not been implemented because of fundamental questions that still exist regarding feature-based operations:

- What are the editing operations that best express the designer's vocabulary?

- What operations are more popular, important and appropriate to a specific domain?
- How are operations to be applied whilst maintaining the validity of the model?
- What are the consequences of implementing new operations, i.e. do they require further supporting operations or other revalidation operations?
- What interdependency exists between these feature-based editing operations?

It was thus found that further research needs to be done on the issues of defining and implementing feature-based editing operations.

10.7 THE REASONING PRIORITY

The rules were initially implemented without any concern for their priority arrangement. This was done in an attempt to use a CLIPS facility that executes rules in an unpredictable order to achieve its goals.

However, the development was made easier and made more sense when different levels of priorities were used. These levels stabilised with a set of four groups of rules (Figure 7-2) for the conceptual MFI validation. Subsequently, the reasons for their internal relationship was realised through a metaphorical parallel (see section 7.3.3). RDI's were added with the consequent addition of more reasoning sets (Figure 7-3) but following the same (linguistic-like) arrangement.

This possibly means that some sort of organisation of the types and levels of validation is a requirement for a coherent system implementation. Another way to obtain and justify a similar priority arrangement could have been through a

study of the process designers adopt to perform these same analyses but in a manual (although computer-aided) way.

During the process of developing the rules to accommodate the MFI's and RDI's, an experimentation with different possible priority organisations of the reasoning sets was carried out. Other arrangements that were used seem to have worked for some tests but the one presented has worked for all tests. Although a more elaborate test process can be performed, it is believed that the ones performed suffice for the prototype implementation in order to validate the validation framework. Moreover, the result of this experimentation produced a priority arrangement that fits a metaphorical parallel to another type of analysis (linguistic) and this is considered an reinforcement of the validity and suitability of the priority arrangement.

10.8 INTENTS MANAGEMENT

Because design is an operation that generates temporary inconsistent states, an intent-driven DbF system should have automatic verification and enrichment reasonings that create and delete FbDI relationships, as they occur.

Verification and enrichment statements are the means by which intent-driven validation is achieved and implemented. However, not all FbDI's have both verification and enrichment statements. Enrichment statements in particular are hard to conceive and, although possible in some cases, they are often not practical. For instance, Parametric FbDI's are composed essentially by verification statements (in a cyclic process of deriving parameter values and checking them against constraints).

Morphological functional conceptual validation is composed mainly of verification statements while RDI validation makes intensive use of enrichment statements.

Despite the fact that the aim of devising a system with PDI enrichment (automatic constraint recognition) has been established (Mantylä94) this seems to be an impractical task when viewed from a more comprehensive perspective. However, narrower attempts to perform some sort of parametric enrichment have been made by Suzuki et al. (1990) for under-constrained 2D parametric designs and by Silva et al. (1990) for positional and orientational GDI's.

10.9 THE IMPLEMENTATION

It can be asserted that if the rule execution sequence is not important for a knowledge-based system (KBS) implementation and the same result is achieved regardless the order, then a more pre-determined sequence of execution should also achieve the same result.

Furthermore, KBS systems, including CLIPS, offer the possibility of defining knowledge partitions (i.e. modules or groups of rules) and therefore, a pre-determined sequencing such as the one defined in FRIEND's reasoning (section 7.6) does not contradict the KBS paradigm (unless expanded towards sequencing the rules themselves instead of groups of rules).

CLIPS version 6.0 was used in an effort to understand the KBS technique. However, its potential (alternative solution search mechanisms) was not fully explored and, in fact, the prototype system could have been implemented using a standard C or C++ language. It did prove though, to be an easy and powerful language/system to work with, especially because of the multi-programming paradigm characteristic that allows procedural, object-oriented and rule-based programming to be used wherever it seems most appropriate.

It was found that the *active* and *passive* characteristics of the feature interaction case (e.g. *enter* or *entered*; *cross* or *crossed*) did not help simplify the reasoning and they could always be converted into an active behaviour for both volumetric (VI) and boundary (BI) cases.

The same can not be said about FI cases because the *master* and *slave* of the RELATION had already been established at a higher level (VI or BI RELATION that points to the FI RELATION). Thus, FI cases are still classified into *active* and *passive* cases (e.g. *inside* or *outside*). BI *active* and *passive* cases were also unified because they originated from conjoint VI cases only and thus were implemented as a specialisation of VI cases (and not as a link to another INTERACTION object) and thus they could be simplified (inverted).

The implementation of the FRIEND-VIEW module, i.e. the visualisation of the part description file (".prt"), using Microstation BASIC language has not been described because it was considered almost trivial for the following reasons: (a) a communication file was used as an interfacing technique, (b) the geometric operations required by the reasoning were implemented within CLIPS; (c) the geometric simplicity of the feature taxonomy adopted fitted the geometric primitive solids available in the MODELER module, and (d) the emphasis of this work was on the reasoning process and capabilities of the validation analysis rather than on features' internal low-level representation issues.

11. CONCLUSION

11.1 SUMMARY OF THE THESIS

This thesis can be summarised through the following main accomplishments:

- A better understanding of the problems involved in the development of a feature-based modelling system such as the model editing, DbF and FeR approaches integration and the reasoning and representation of feature models.
- A comprehensive review and study of the literature concerned with feature-based modelling has been presented. In particular this covers: feature-based modelling and systems, feature-based interaction problems and determination and validation analysis.
- The identification of the validation problem as an important aspect for existing systems as well as the lack of a more detailed literature on this subject. It has been alleged that many systems support some kind of validity checking but details of such functionality are most often omitted.
- A novel framework for implementing design-by-features systems where the validation of a feature-based model representation is the central concern.

- The determination and use of a methodology, resembling the feature *elicitation process*, to help specify the components of the validation framework.
- The production of original classifications and taxonomies for feature-based designer's intents (FbDI's) and feature operators.
- The development of a methodology to identify and classify feature-based interaction cases based on Boolean operators and simple queries to the GSM representation. This feature-based interaction classification is independent of possible interpretations of each case therefore could be applied to other systems.
- The development and codification of algorithms and production (IF-THEN) rules to express FbDI's, identify feature interactions and perform feature operations.
- The development and successful implementation of the conceptual reasoning after observing that distinct aspects drive the *morphological functional* FbDI (MFI) reasoning and these are responsible for the priority arrangement which was presented.
- The generalisation of the MFI reasoning towards the intent-driven approach through outlines of *enrichment* and *verification* rules and a management strategy was explained. The statuses of the entities used in this reasoning and a possible interpretation of the identified priority were also presented.

- A distinctive working prototype system, implemented from scratch, using the proposed framework and taxonomies, which is capable of:
 - ⇒ Modelling feature-based parts and the monitoring of FbDI's and features in the model.
 - ⇒ Maintaining the model's validity by reasoning with the model and suggesting or applying selective corrections.
 - ⇒ Understanding enriched and complex feature-based models, by identifying and representing features, interactions and FbDI's.
 - ⇒ Reacting to the effects of manipulating features and FbDI's.
 - ⇒ Recognising implicit intents originated through the design-process via *enrichment* statements.
 - ⇒ Capturing, representing and manipulating FbDI's and feature interactions explicitly.

11.2 CONTRIBUTIONS

It is believed that this work has contributed to knowledge in the following aspects:

11.2.1 CLARIFICATIONS

The nature of the validation problem has been clarified through the identification, definition, classification, specification and devising of a solution for feature-based representation validation.

Within this clarification, the roles of geometric validation and feature-based representation validation have been established and the terms feature-based designer's intents, feature interaction and validation have been formally defined.

A taxonomy of FbDI's has been created to help in clarifying features themselves from the perspectives of their intended semantics, expected behaviour and application. This clarification, and indeed this taxonomy, has been used as a framework for validating feature-based models.

A contribution has also been made in clarifying and distinguishing some of the information involved in such a system. For instance, some confusion was found in the literature between the terms feature interaction and feature interference, between interaction and intents, and between FbDI's and features themselves.

11.2.2 EFFECTIVE CAPTURE OF DESIGNER'S INTENTS

The clarification and formal distinctions between the concepts involved, has allowed a clearer understanding of their extent, and this has made it possible for this prototype implementation and future full implementations to represent and capture them distinctively.

The research demonstrates that it is possible for a FBM system to effectively and explicitly represent, capture, manipulate and use designer's intents for reasoning during on-going design. A clear way of defining, identifying and analysing valid and invalid model representations based on FbDI's has also been presented.

11.2.3 THE VALIDATION FRAMEWORK

A novel approach for the development of a DbF system centred on the concepts of representation validation and FbDI management has been presented. The contents of a valid feature-based model representation have been defined, and methods to analyse and maintain the validity of the representation according to the features' expected behaviours and intents of have been demonstrated.

- The elements required for such a validation system have been presented, comprising Information (features and FbDI's), Operations, Interrelations (FbDI and interaction representations) and an architecture (reasoning sets and a priority arrangement).
- The interaction of features and their mutual effect on validity have been studied and implementation methodologies presented (section 7.2). The

validation framework also provides an important mechanism for remedying invalid representations through the selective application of *revalidation* operations.

- The feasibility of the validation framework in validating the representation and applying a formal but simple validation formalism for developing FBM systems has been exploited. The implementation has shown that the validation of the feature's concept can be made partially at the feature class level. Feature type reasoning starts to populate the analysis as soon as it goes further in understanding the model.
- The validation framework organises and integrates the information required for the validation reasoning. Helped by the priority arrangement, intent management statements and the specification of the information, the framework binds the information and gives a complete structure for a system capable of performing intent-driven conceptual feature-based representation validation.
- A major contribution of this study is the novel methodical arrangement (or structuring) of information used in feature-based modelling in terms of:
 - a formal implementation via a validation-based approach;
 - components and their definition/use/binding (interaction cases, operations and designer's intents);
 - the detailing of the components (the application of the *elicitation process*);

- defining groups of distinct reasonings;
- the organisation of the components (priority and management);
- what can be achieved with intent-driven validation environments.

Figure 11-1 shows the structures that have been identified.

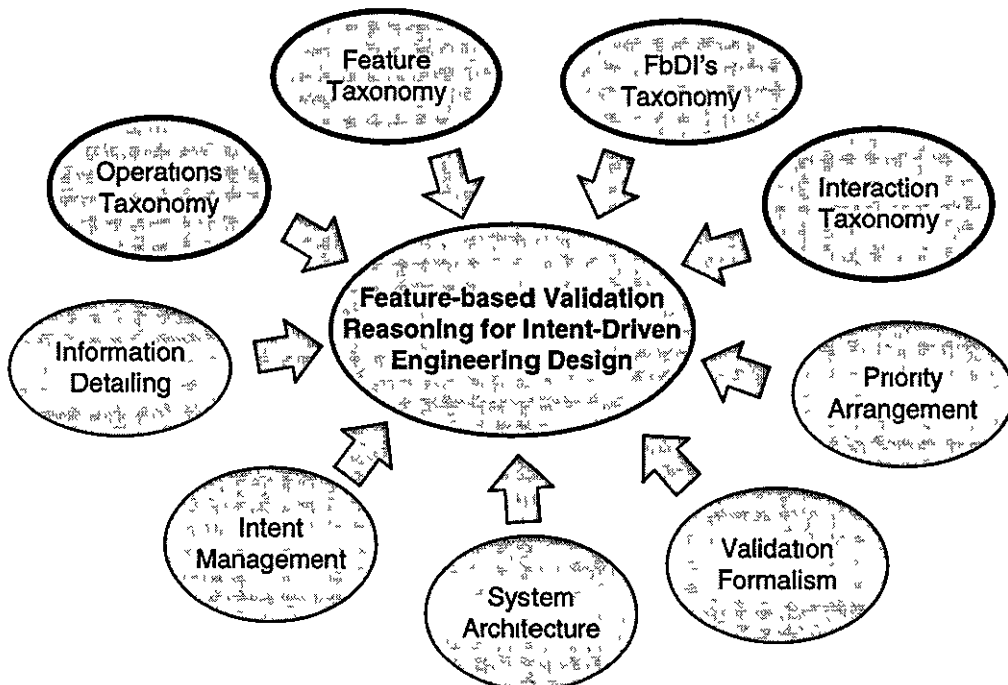


Figure 11-1: Structures Involved in an Intent-driven Validation System.

11.2.4 INFORMATION DETAILING (THE PROCESS)

The application of the entity *elicitation process* (a loose formalism), contributed to the detailing of the information required for the intent-driven validation system. Classifications and taxonomies for the individual elements were an important outcome of this process. The application of this formalism was done in such a way that not only was the information properly identified but also the distinction between types of information was emphasised.

It is believed that this formalism could be applied to other types of information and the FBM system could grow to comprise the central core of Product Data Modelling (PDM) techniques. This future system would have taxonomies for information such as features, operations and processes, in a similar way to that implied in Mäntylä97.

11.2.5 CLASSIFICATIONS AND TAXONOMIES (THE PRODUCTS)

In applying the *elicitation process*, it has been possible to classify and give names to various items for each information type involved.

Classifications are important because they help to group properties and to highlight differences. In particular they help emphasise the completeness of the subject and in so doing help to identify the absence of elements.

The taxonomies for feature-based designer's intents, feature interaction cases and feature operations are singular and each one is a useful contribution *per se*.

The taxonomy of feature-based designer's intents (FbDI's) establishes (Figure 4-10) morphological, theoretical and relational functional designer's intents as the highest level. Morphological functional FbDI's are defined as volumetric relationships such as fittability, changeability, labelling and redundancy. Theoretical functional FbDI's are considered related to parametric constraint-based design approaches. Relational functional FbDI's (RDI's) are subdivided into geometric RDI's and application-oriented RDI's. Geometric RDI's are positional, structural, orientational, and hierarchical intentions while application-oriented RDI's identified include temporal, precedence, compound and proximity intentions.

The taxonomy of feature interaction cases establishes (Figure 5-13) a classification framework that is applied at three different levels of the geometric representation of features. This framework identifies features that are connected or disconnected. Disconnected features can be adjoint or disjoint

while connected features can be conjoint or subjoint. These can be further detailed to obtain interaction cases that include features with matching faces, features crossing each other, a feature near or far from another, and so on.

The feature interaction classification identification (Figure 5-9) is the unique product of a well-defined and specified methodology. Moreover, although its usefulness and feasibility were partially experienced in **FRIEND**, it is a complete and deep research topic that deserves further research.

The taxonomy of feature operations (Figure 6-2) distinguishes between analysis, manipulation and setup operations. These operations can be further sub-classified according to the type of information involved (i.e. FbDI's, the GSM representation or to the FBM representation). Manipulation operations include editing, modelling and revalidation operations. Editing manipulations are of either high (applied onto the FBM representation) or low level (applied to the GSM representation).

11.2.6 PRIORITY ARRANGEMENT

It has also been possible to identify types of rules, intent management statements and how they can be implemented. Although not final (further FbDI development would possibly require adaptations) and not definite (alternative schemes can be invented), the reasoning priority (Figure 7-6) for all the FbDI's involved in this research is a significant contribution because it establishes an arrangement that forms the basis for discussion and for extensions related to the future development of FbDI's.

11.2.7 ADDED DEVELOPMENT FORMALISM

A few formal architectures, approaches and methodologies for FBM system development have been found in the literature (Gandhi89, Subrahmanyam95, Hailong95, Kim96) but they are in their infancy despite the fact that many systems have already been implemented.

The borrowing of methods (feature *elicitation process*) and mimicking of ideas (linguistic analysis) is believed to have added an extra formalism for the development of FBM systems and, although these have not been presented as such, they could be developed further in order to originate a complete development formalism and methodology.

11.2.8 A COMPARISON FRAMEWORK

Feature based modelling systems and applications are usually compared through the extent of the feature taxonomy they can represent together with the analysis or synthesis capabilities of the specific application involved (such as finite-element analysis, setup planning and tooling planning).

This work contributes in offering other elements to extend this comparison. FBM systems can now be compared by the intelligence and behaviour of features (reasoning capabilities), the feature interaction cases supported, the operations available and by the range of feature-based designer's intents that can be represented and reasoned with.

11.3 FURTHER WORK AND FUTURE RESEARCH

This thesis raised some issues that could be divided into two streams: (a) those that further develop the existing ideas towards a more practical implementation, which would most probably only consist of implementation issues, and are thus called extensions, and (b) those that represent future research in this area, which involves the exploration of related ideas, and are called expansions.

11.3.1 EXTENSIONS

The purpose of this research was not to produce a fully implemented fully working system for practical use but to explore some fundamental issues. Therefore, there is a possibility of re-implementing some details in order to produce a more practical system:

- A more efficient implementation.

Most of the calculations and functions were built within the CLIPS environment (even some Boolean operations and bounding box analysis). It is believed that a better integration between the GSM and the KBS environments could produce a more efficient implementation and add some useful graphical manipulation capabilities. In addition, some bounding box tests would then be verified by the GSM module and therefore, the interaction identification would be more accurate.

- Analyse features of additive *nature*.

Although the stock material was considered and implemented as a *satellite* feature (see taxonomy of features adopted in this work in Figure 1-6), of positive *nature*, further analysis is required for other positive features in the model, such as *bosses* and *ribs*.

This would lead to the need for more tentative exploration (possibly through the use of all KBS capabilities) of revalidation scenarios (especially when deploying the *split* and *merge* revalidation operations) based on some criteria such as setup or tooling access.

- Hierarchical modelling.

System functionality can be improved and implementation facilitated if a hierarchical *parent-child* relation is imposed. The current implementation of **FRIEND** does not force a hierarchical structure among features. However, this mechanism facilitates parametric relationships between features and could therefore it could help to add this level of validation. Nevertheless, modelling hierarchies and hierarchies of features defined by attributes or FbDI's are distinct items of information that should be clarified and approached in ways yet to be defined.

- Thorough extension towards process planning.

It is considered that FbDI's and operations presented here are particularly valid for CAPP applications because they were gathered from research reports mainly on CAPP and Design-for-Manufacturing feature-based systems. This should be verified and a more reliable association between FbDI-operations-application could be achieved.

To define the extent of the FbDI's usefulness for facilitating (if not automating) process planning generation it would be interesting to see a CAPP system thoroughly using FbDI's with few, if any, enquiries to a low-level geometric reasoning system.

- A better *intent*.

A better internal representation of FbDI's would allow some references and/or values to be stored with the intent. This could possibly be achieved through the use of *frames* to accommodate parameters and functions as part of the intent data-structure.

A better FbDI that relates various features would allow the generation of an intricate mesh of relationships which raises the question of how to simultaneously present such a plethora of FbDI's to the user in order to allow their visualisation and management.

11.3.2 EXPANSIONS

The work presented in this thesis represents a Product Data Modelling (PDM) approach aimed at producing a better (guaranteed correct and enriched) representation of a part. The expansions proposed below follow different paths in fulfilling the modelling of information required for a PDM approach where it is believed that feature-based intent-driven validation is capable of providing a bridge between rigid and flexible information embedded in the design task.

- Expansion of the reasoning capabilities of **FRIEND** to accommodate various applications (e.g. tolerancing, setup planning, fixturing) to study the concurrency problem alongside FbDI's and to investigate how the validation framework would respond to this new requirement.

Instead of a linear declarative reasoning, this more concurrent approach would probably require the use of another AI technique. A "blackboard system" approach seems more appropriate to perform this reasoning task.

This also would require a taxonomy that classifies FbDI's not according to their functional meaning but according to expertise or responsibility. For

instance, VDI's could be assigned to a designer, PDI's assigned to a structural analysis engineer and RDI's to a process planning engineer.

- *General propagation* poses the problem of identifying the variety of possible alternative operations to propagate. For instance, *stretching* the thickness (or height) of the part in Figure 11-4, where the *holes* were defined as *nested@bot* of the *step* feature produces a plethora of alternative propagation operations (see Figure 11-2):

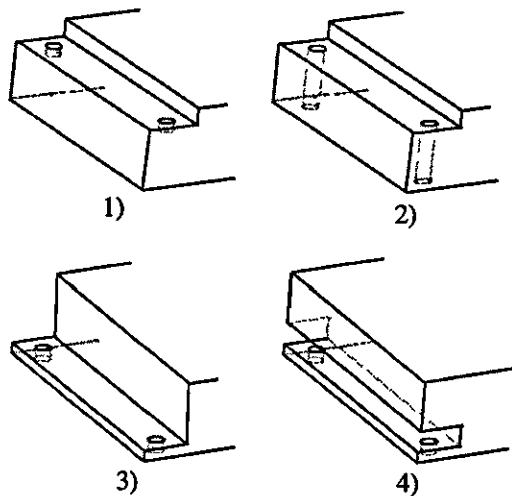


Figure 11-2: Alternative Revalidation Operations

1. Change the *step* positioning (keep *step volumetric intention*); change *hole* positioning and labels to *blind-holes* (keep *hole volumetric intentions*);
2. Change the *step* positioning (keep *step volumetric intention*); increase the *hole* height;
3. Change the *step volumetric intention*; Keep *step* positioning (keep *hole volumetric intention* and positioning);

4. keep *step volumetric intention* and positioning; Change the *step* label to a *slot_tru* (keep *hole volumetric intentions* and positioning);
5. Invalidate height manipulation because of the functional significance of the *hole/step* arrangement.

A possible approach to tackle this problem would be to search all features affected by the operation and their properties and ask the designer which are to be kept and which are to be changed. This approach however, introduces no “intelligence” to the system. A study of possible scenarios could result in a more guided, useful and intelligent (reasoned) set of alternatives/suggestions. An integration of existing FbDI’s with constraint-driven systems could facilitate the determination, and even derivation, of the more likely and correct alternatives.

- One of the problems of form-function relationships is the multiplicity of forms that could be used to implement a target function. The validation mechanism could be expanded upwards in the abstraction level and (high-level) *abstract FbDI’s* defined. Such a system would track the validity of (low-level) FbDI’s with the possibilities supposedly associated with a (high-level) *abstract FbDI* - a function

In this way, even if a totally different FbDI scenario is achieved after an operation, it could still be valid because it fulfils one of various possible scenarios that satisfy the implementation of an *abstract FbDI*.

It could be useful to save *abstract FbDI’s* as a blue print of that part and see how this knowledge could be reused in order to design parts that keep some resemblance in functional terms.

- Integration of existing FbDI’s with Parametric FbDI’s, i.e. a constraint-driven approach. Although different validation capabilities will be

achieved, these should be analysed in conjunction with the ones presented here because PDI's (constraints) were not considered to be directly included in the proposed reasoning priority arrangement.

The main benefit of FbDI's relate to the flexibility of the approach compared to a constraint-driven-only approach. In this sense, all attributes of a model (e.g. assembly, tolerances) that could be categorised as flexible (*unbound*) or fixed (*tightly-bound*) behaviour could benefit from being studied and included in the FbDI's taxonomy. Attributes with *fixed* behaviours could be defined as those that are assigned to the model to conform to some requirement while attributes with *flexible* behaviours could be those that are included in the model due to completeness requirements.

A possible solution would be to record this "flexibility" behaviour as a FbDI parameter, i.e. it would be interesting to have the possibility to assign behaviours such as *unbound*, *loosely-bound* or *tightly-bound* to determine how a system would react to changes in the FbDI relationship. For instance, an *unbound* FbDI could be created and deleted regardless of the explicit consent of the user while a change to a *tightly-bound* FbDI could only be carried out after being confirmed by the user. This has some similarities to the proposal by Nielsen et al. (Nielsen91).

FbDI's help model (loose) relationships that, possibly via other (loose) relationships, could be related to a more tightly-defined constraint. A different co-ordination between PDI's and the other FbDI's is thus required in order to not lose the flexibility of an intent-driven approach.

- Different views of a product could produce not only different feature-based models but also different associated FbDI perspectives. This raises the problem of mapping FbDI's across different views. Although this is a problem in itself, FbDI mapping could facilitate the mapping of features

across feature spaces and facilitate the understanding of the component's requirements from another perspective. For instance, a process planning engineer trying to change a parameter could receive messages that this violates a set of FbDI's when mapped to another, say production planning, view.

If the *flexibility* attribute defined earlier were already implemented, those messages could be:

- *warning* messages for both views (process and production planning engineers for the example given) if *unbound* FbDI's are involved;
- *impediment* messages if *tightly-bound* FbDI's are involved (and possibly the automatic issuing of a design change request if the FbDI is also *tightly-bound* in the other view);
- if the corresponding FbDI in a view "B" is of a *loosely-bound* type, the required changes in a view "A" would not be prevented but the user working on the view "B" would be warned of the changes in the view "A" (and possibly, how they are mapped into view "B").

In this way, FbDI mapping could greatly enhance the negotiation of information related to a product from various perspectives.

- At the same time, this FbDI integration could help minimise the search space of possible alternatives for *general propagation*. A balance would have to be achieved between *unbound* (FbDI's) and *tightly-bound* designer's intents (constraints). Because some reasoning and interactions could be done using bounding boxes, this may be suitable for relaxation reasoning (Mäntylä89, Case92b, Das96) where some of the properties of a feature (such as its *volume* or orientation) are left unspecified (*fuzzy* or *relaxed*) and the system would reason about these properties using the

FbDI's generated during design against cost/time criteria (for example) and suggest alternative designs.

Figure 11-3 presents the relaxation group of a *slot* feature.

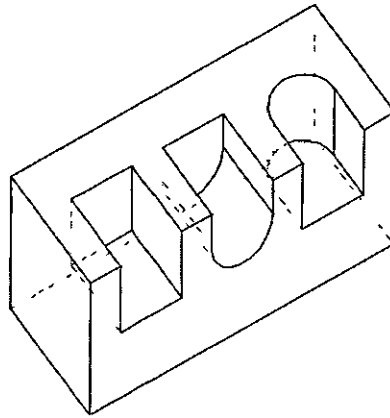


Figure 11-3: Alternative Relaxations.

The idea of relaxation can be extended to produce multi-layered models that present or hide details from the user, through *refinement* or *relaxation* processes (Figure 11-4) according to need and perspective while storing some of the history of the design. History of the design has proved to be an important by-product of the design process (Rossignac90, Sreevalsan92, Rosen93).

For instance, an abstract layer of the feature-based model could be used to produce fast animations while another (possibly more detailed) model could be used by the designer by removing some of the relaxation degrees of freedom. Furthermore, the process planning engineer would receive the model with all information (or suggestions) for the remaining relaxation degrees of freedom.

Figure 11-4 shows a part being modelled in various layers. The highest layer is the most abstract one and outlines the part via its bounding box

dimensions. Subsequent layers are refinements of higher layers and produce lower abstraction layers with more detailed (less relaxed) information of the part. The refinement process has a counterpart which is the relaxation process. A relaxation process would make the details of a feature back to a relaxed (less refined) form. Alternative realisation of the part could be achieved from different refinements from the relaxed form.

It is considered that a PDM technique is concerned with modelling various types of information regarding a product that include:

- modelling the *geometry* of the product where a validated feature-based representation and a solid representation derived from it are the most important elements;
- modelling the *knowledge* associated with the product, where (low-level) FbDI's (including *unbound*, *loosely-bound* and *tightly-bound* FbDI's) and feature interactions can be an important part of the knowledge alongside the modelling of processes and the application's restraints associated with the product.
- modelling the *design process* used to generate the product, where feature operations, (high-level) *abstract FbDI's* and relaxed/refined layered representation could enhance the history associated to the product modelling.

It can be inferred through the expansions discussed above that, as a Product Data Modelling (PDM) technique, a feature-based intent-driven validation system could represent a valuable development framework.

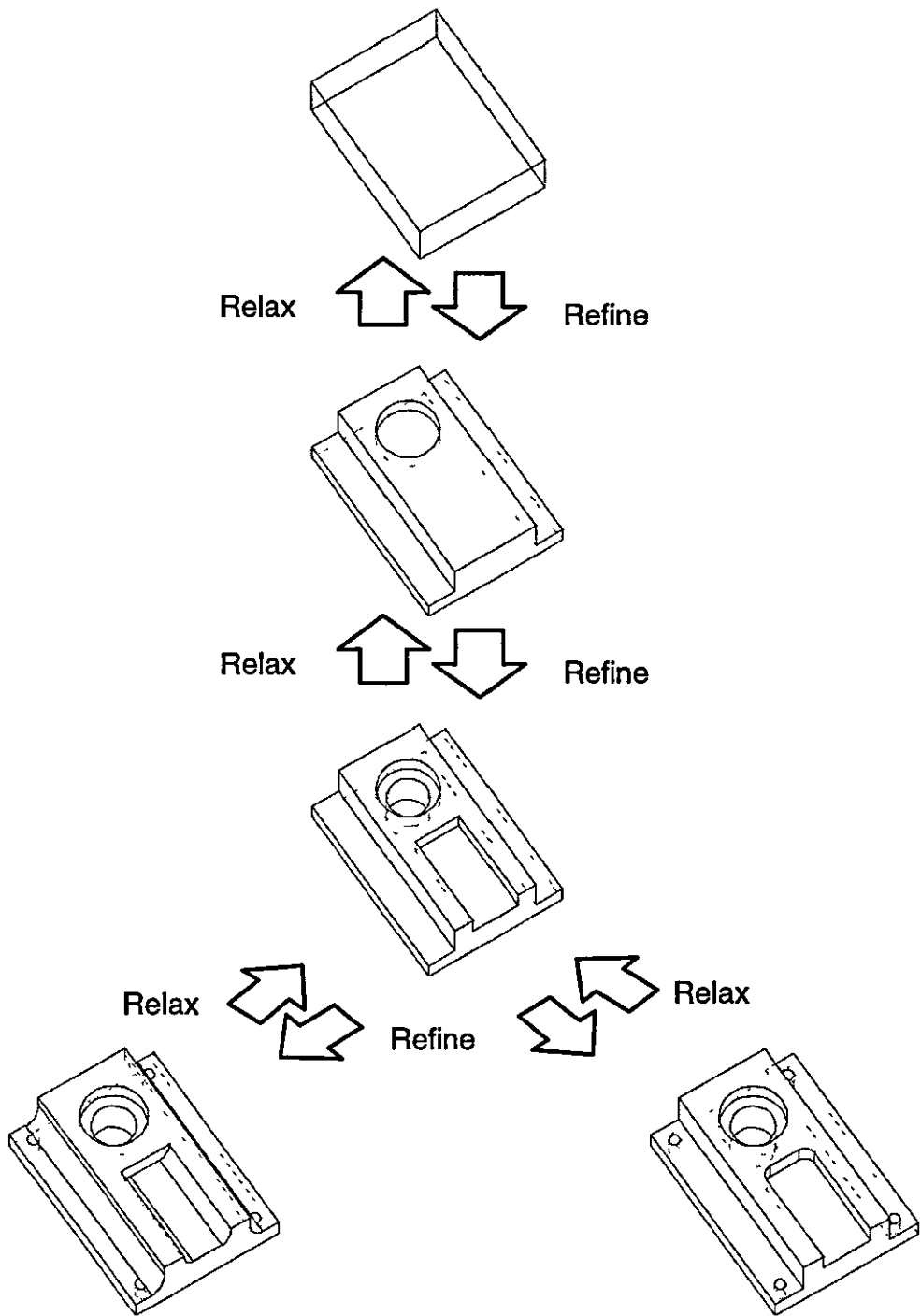


Figure 11-4: Multi-Layered Feature-based Relaxed Model.

11.4 FINAL REMARKS

To summarise the contributions of this work, it can be asserted that a novel approach for DbF systems development was presented, information structures were identified and specified, formalisms were applied and prioritisations were established towards a feature-based intent-driven validation system.

The prototype system **FRIEND** (Feature-based validation Reasoning for Intent-driven **EN**gineering Design) has a clear validation procedure (because its major concern is the validity of the model) and explicitly captures designer's intents (which are not geometric constraints) because it is driven by intents reasoning.

It is claimed that the ideas presented here have contributed to allow the development of a FBM system that would give a better support for detailing a geometric design by giving feedback on the validity of the model regarding well-defined properties of feature themselves as well as their application characteristics.

The designing task would be made easier because of the support of an underlying intelligent and **FRIEND**ly reasoning capable of understanding FbDI's and helping to produce an enriched and valid model.

It is also claimed that the system development task would be greatly facilitated by considering the formalisms and frameworks presented here as guidelines and the classifications and taxonomies for comparison and discussion.

It is hoped that this work would, at least, help raise the awareness of the validation problem in future feature-based modelling system developments (especially DbF systems) and provide a initial solution framework for discussion.

12. REFERENCES

- (Allada95) Allada, V. and S. Amand. "*Feature-based Modelling Approaches for Integrated Manufacturing: State-of-the-art Survey and Future Directions*". International Journal of Computer Integrated Manufacturing. Vol. 8(6), pp. 411-440. 1995.
- (Anderson90) Anderson, D. C. and T. C. Chang. "*Geometric Reasoning in Feature-Based Design and Process Planning*". Computers and Graphics. Vol. 14(2), pp. 225-235. 1990.
- (Bidarra93) Bidarra, R. and J. C. Teixeira. "*Intelligent Form Feature Interaction Management in Cellular Modeling Scheme*". (ACM/IEEE) Second Symposium on Solid Modeling and Applications, May 19-21, Montreal, Canada. Vol. 1, pp. 483-485. 1993.
- (Bidarra94) Bidarra, R. and J. C. Teixeira. "*A Semantic Framework for Flexible Feature Validity Specification and Assessment*". (ASME) International Computers in Engineering Conference and Exhibition. Vol. 1, pp. 151-158. 1994.
- (Bidarra96) Bidarra, R. and W. F. Bronsvort. "*Towards Classification and Automatic Detection of Feature Interactions*". Proceedings of the 29th ISATA Conference; Mechatronics - Advanced Development Methods & Systems for Automotive Product, Florence, Italy. Vol. 1, pp. 99-108. 1996.

- (Bronsvoort93)** Bronsvoort, W. F. and F. W. Jansen. *"Feature Modelling and Conversion - Key Concepts to Concurrent Engineering"*. Computers in Industry. Vol. **21(1)**, pp. 61-86. 1993.
- (Butterfield85)** Butterfield, W. R., M. K. Green, D. C. Scott and W. J. Stocker. *"Part Features for Process Planning"*. CAM-I Inc, Arlington, Texas, USA. Vol. **1**, 1985.
- (Case92a)** Case, K. *"Feature Technology - an Integration Methodology for CAD and CAM"*. International Conference on Manufacturing Automation, Hong-Kong. Vol. **1**, pp. 613-624. 1992.
- (Case92b)** Case, K. *"Feature-based CAD Systems for Process Capability Modelling"*. International Congress on Computer Graphics-CICOMGRAF'92, São Paulo, Brazil, 30th . July. Vol. **1**, pp. 1-14. 1992.
- (Case93a)** Case, K. and J. Gao. *"Feature Technology: An Overview"*. International Journal of Computer Integrated Manufacturing. Vol. **6(1-2)**, pp. 2-12. 1993.
- (Case93c)** Case, K., J. Gao, and N. N. Z. Gindy. *"LUT-FBDS: A Feature-based Design System"*. Final Report, SERC Grant GR/G35657, Loughborough University of Technology. 1993.
- (Case94)** Case, K., J. X. Gao, and N. N. Z. Gindy. *"The Implementation of a Feature-based Component Representation for CAD/CAM Integration"*. Journal of Engineering Manufacture - Proceedings of the Institution of Mechanical Engineers (IMechE) Part B.. Vol. **208(B1)**, pp. 71-80. 1994.
- (Chamberlain93)** Chamberlain, M. A. *"Protrusion-features Handling in Design and Manufacturing Planning"*. Computer Aided Design. Vol. **25(1)**, pp. 19-28. 1993.
- (Chang90)** Chang, T.-C. *"Expert Process Planning for Manufacturing (ISBN 0-201-18297-1)"*. Addison-Wesley Publishing Company, Reading, USA. 1990.

- (Chen94) Chen, C. L. P. and S. R. LeClair. *"Integration of Design and Manufacturing: Solving Setup Generation and Feature Sequencing Using an Unsupervised-Learning Approach"*. Computer Aided Design. Vol. 26(1), pp. 59-75. 1994.
- (Chen95) Chen, X. and C. M. Hoffmann. *"On Editability of Feature-based Design"*. Computer Aided Design. Vol. 27(12), pp. 905-914. 1995.
- (Choi84) Choi, B. K., M. N. Barasan, and D. C. Anderson. *"Automatic Recognition of Machined Surfaces from 3D Solid Modelling"*. Computer Aided Design. Vol. 16(2), pp. 81-86. 1984.
- (Chovan91) Chovan, J. D. and M. B. Waldron. *"Identifying the Fundamental Geometric Attributes of Design: an application of distinctive feature theory"*. 3rd. International Conference on Design Theory and Methodology (ASME-DE). Vol. 31, pp. 209-216. 1991.
- (Chung90a) Chung, J. C. H., D. R. Patel, R. L. Cook and M. K. Simmons. *"Feature-Based Modelling for Mechanical Design"*. Computers and Graphics. Vol. 14(2), pp. 189-199. 1990.
- (Chung90b) Chung, P.W.H., et al. *"Chapter 9: Overview of Artificial Intelligence Tools" in Knowledge-Based Systems for Industrial Control*, Peter Peregrinus Ltd. pp. 165-188. 1990.
- (Chung90c) Chung, J. C. H. and M. D. Schussel. *"Technical Evaluation of Variational and Parametric Design"*. (ASME) Computers in Engineering. Vol. 1, pp. 289-298. 1990.
- (Collins87) *"Collins COBUILD English Language Dictionary (ISBN: 0-00-370023-2/3)"*. Developed and Compiled in the English Department at the University of Birmingham. HarperCollins Publishers. 1987.
- (Crawford93) Crawford, R. *"Integrating 3D Modelling and Process Planning by Features: a Case Study"*. International Journal of Computer Integrated Manufacturing. Vol. 6(1-2), pp. 113-118. 1993.

- (Cunningham88)** Cunningham, J. J. and J. R. Dixon. *"Designing with Features: the Origin of Features"*. ASME Computers in Engineering Conference. Vol. 1, pp. 237-243. 1988.
- (Das96)** Das, D., S. K. Gupta, and D. S. Nau. *"Generating Redesign Suggestions to Reduce Setup Cost: A Step Towards Automated Redesign"*. Computer Aided Design. Vol. 28(10), pp. 763-782. 1996.
- (Denzel93)** Denzel, H. and G.-C. Vosniakos. *"A Feature-based Design System and Its Potential to Unify CAD and CAM"*. Interfaces in Industrial Systems for Production and Engineering (IFIP-Transactions B: Applications in Technology), Elsevier Science Publishers B. V. (North-Holland). Vol. B-10, pp. 131-144. 1993.
- (Dixon87)** Dixon, J. R. and J. J. Cunningham. *"Research in Designing with Features"*. IFIP WG 5. 2, Conference on Intelligent CAD, Boston. Vol. 1, pp. 137-148. 1987.
- (Dixon90)** Dixon, J. R., E. C. Libardi Jr, and E. H. Nielsen. *"Unresolved Research Issues in Development of Design-with-Features Systems"*. IFIP WG 5. 2/ NSF Working Conference on Geometric Modelling, Wozny, M. J., Turner, J. V. and Preiss, K Editors. RensselaerVille, USA, Elsevier Science Publishers B. V. (North-Holland). Vol. 1, pp. 183-196. 1990.
- (Dohmen94)** Dohmen, M. *"Constraint Techniques in Interactive Feature Modeling"*. Report 94-16, Delft University of Technology (TUDelft), Faculty of Technical Mathematics and Informatics. 1994.
- (Dohmen96)** Dohmen, M., K. J. d. Kraker, and W. F. Bronsvort. *"Feature Validation in a Multiple-View Modelling System"*. (ASME) Design Engineering Technical Conference and Computers in Engineering Conference, Irvine, California, USA. Vol. DETC-96, pp. 1-10. 1996.
- (Donnell94)** Donnell, B. L. *"Object/Rule Integration in CLIPS"*. Expert Systems. Vol. 11(1), pp. 29-45. 1994.

- (Duan89) Duan, W., W. Qifu, and J. Zhou. *"The Research of Feature Solid Modelling"*. 5th International Conference on Computer-Aided Production Engineering, Edinburgh, Scotland, UK. November. Vol. 1, pp. 391-396. 1989.
- (Duan93) Duan, W., J. Zhou, and K. Lai. *"FMST: A Feature Solid Modelling Tool for Feature-Based Design and Manufacture"*. Computer Aided Design. Vol. 25(1), pp. 29-38. 1993.
- (Eastman84) Eastman, C. M. and K. Preiss. *"A Review of Solid Shape Modeling Based on Integrity Verification"*. Computer Aided Design. Vol. 16(2), pp. 66-80. 1984.
- (ElMaraghy91) ElMaraghy, H. A. *"Intelligent Product Design and Manufacture"*. In *Artificial Intelligence in Design*, D. T. Pham, Editor. Springer-Verlag. pp. 147-168. 1991.
- (ElMaraghy93a) ElMaraghy, H. A. *"Evolution and Future Perspectives of CAPP"*. Annals of the CIRP. Vol. 42/2, pp. 739-751. 1993.
- (ElMaraghy93b) ElMaraghy, H. A., K. F. Zhang, and H. Chu. *"A Functional-Oriented Modeler Prototype"*. (ASME) Design Engineering Conference (Conference Code 18671) ISBN:0-79-181136-0, Chicago, USA, March. Vol. DE-52, pp. 57-62. 1993.
- (Emmerik89) Emmerik, M. J. G. M. v. and F. W. Jansen. *"User Interface for Feature Modelling"*. (IFIP) Computer Applications in Production and Engineering Conference, Elsevier Science Publishers B. V. (North-Holland). Vol. 1, pp. 625-632. 1989.
- (Emmerik91) Emmerik, M. J. G. M. v. *"Interactive Design of 3D Models with Geometric Constraints"*. The Visual Computer. Vol. 7(5-6), pp. 309-325. 1991.

- (Faux86) Faux, I. D. "Reconciliation of Design and Manufacturing Requirements for Product Description Data Using Functional Primitive Part Features". Report R-86-ANC/GM/PP-01.1, CAM-I Inc., 1986.
- (Fu94) Fu, Z. and A.Y.C. Nee. "Interpreting Feature Viewpoint to Concurrent Engineering". (ASME) International Computers in Engineering Conference and Exhibition, Minneapolis, Minnesota, USA, September, 11-14. Vol. 1, pp. 405-411. 1994.
- (Gadh95a) Gadh, R. and F. B. Prinz. "A Computationally Efficient Approach to Feature Abstraction in Design-for-Manufacturing Integration". Journal of Mechanical Design. Vol. 117(February), pp. 16-27. 1995.
- (Gadh95b) Gadh, R. and F. B. Prinz. "Automatic Determination of Feature Interaction in Design-for-Manufacturing Analysis". Journal of Mechanical Design. Vol. 117(March), pp. 2-9. 1995.
- (Gandhi89) Gandhi, A. and A. Myklebust. "A Natural Language Approach to Feature Based Modeling". (ASME-DE) 15th Design Automation Conference, Montreal, Quebec, CA. September 17-21. Vol. 19-1, pp. 69-77. 1989.
- (Gao93) Gao, J. X. and K. Case. "Information Mapping Between a Feature-based Design and an Integrated Process Planning System". Proceedings of the 28th MATADOR Conference, Macmillan Press. Vol. 1, pp. 551-558. 1993.
- (Giarratano94) Giarratano, J. C. and G. Riley "Expert Systems: Principles and Programming (ISBN: 0-534-93744-6)". 2nd ed. PWS Publishing Company. Boston, USA. 1995.
- (Gindy89) Gindy, N. N. Z. "A Hierarchical Structure of Form Feature". International Journal of Production Research. Vol. 27(12), pp. 2089-2103. 1989.

- (Gindy93)** Gindy, N. N. Z., X. Huang, and T. M. Ratchev. "*Feature-based Component Model for Computer-aided Process Planning*". International Journal of Computer Integrated Manufacturing. Vol. 6(1-2), pp. 20-26. 1993.
- (Gomes91)** Gomes, A. J. P. and J. C. G. Teixeira. "*Form Feature Modelling in a Hybrid CSG/Brep Scheme*". Computers and Graphics (Pergamon Press). Vol. 15(2), pp. 217-229. 1991.
- (Grayer76)** Grayer, A. R. "*A Computer Link Between Design and Manufacture*". Ph.D. Thesis. University of Cambridge. September 1976.
- (Gupta92)** Gupta, S. K., P. N. Rao, and N. K. Tewari. "*Development of a CAPP System for Prismatic Parts Using Feature Based Design Concepts*". International J. of Advanced Manufacturing Technology. Vol. 7, pp. 306-313. 1992.
- (Gupta93)** Gupta, S. K. and D. S. Nau. "*Generation of Alternative Feature-Based Models and Precedence Orderings for Machining Applications*". Second Symposium on Solid Modeling and Applications, May 19-21, Montreal, Canada. Vol. 1, pp. 465-466. 1993.
- (Hailong95)** Hailong, L., D. Jinxiang, T. Min and H. Zhijun. "*A Schema of Developing Feature-based Modelling Systems*". Proceedings of SPIE- The International Society for Optical Engineering (IS:0-81-942012-3). Vol. 2620, pp. 76-80. 1995.
- (Han97)** Han, J. and A. A. G. Requicha. "*Integration of Feature based Design and Feature Recognition*". Computer Aided Design. Vol. 29(5), pp. 393-403. 1997.
- (Harun96)** Harun, W.A.R.J.B.W. "*Feature-Based Representation for Assembly Modelling*". PhD Thesis, Loughborough University, September 1996.

- (Hayes89) Hayes, C. and P. Wright. "Automating Process Planning: Using Feature Interactions to Guide Search". *Journal of Manufacturing Systems*. Vol. 8(1), pp. 1-15. 1989.
- (Henderson90) Henderson, M. R., et al. "Graph-Based Feature Extraction". Arizona State University, USA. 1990.
- (Henderson93) Henderson, M. R. "Representing Functionality and Design Intent in Product Models". Second Symposium on Solid Modeling and Applications, May 19-21, Montreal, Canada. Vol. 1, pp. 387-396. 1993.
- (Herbert90) Herbert, P. J., C. J. Hinde, A. D. Bray, Y. A. Lauenders, D. Round and D. M. Temple. "Feature Recognition within a Truth Maintained Process Planning System". *International Journal of Computer Integrated Manufacturing*. Vol. 3(2), pp. 121-132. 1990.
- (Hummel89) Hummel, K. E. "Coupling Rule-Based and Object-oriented Programming for the Classification of Machined Features". ASME Computers in Engineering Conference and Exposition, Anaheim, California, USA. Vol. 1, pp. 409-418. 1989.
- (Jablokow94) Jablokow, A. G., J. J. Uicken Jr., and D. A. Turcic. "Verification of Boundary Representation of Solid Models". *Journal of Mechanical Design*. Vol. 116(June), pp. 666-668. 1994.
- (Jakiela89) Jakiela, M. J. and P. Y. Papalambros. "Concurrent Engineering with Suggestion Making CAD Systems: Results of Initial User Tests". (ASME-DE) 15th Design Automation Conference, Montreal, Quebec, CA. Sept 17-21. Vol. 19-1, pp. 223-230. 1989.
- (Kang93) Kang, T.-S. and B. O. Nnaji. "Feature Representation and Classification for Automatic Process Planning System". *Journal of Manufacturing Systems*. Vol. 12(2), pp. 133-145. 1993.

- (Kim93) Kim, H. S., H. Ko, and K. Lee. *"Incremental Feature-Based Modeling"*. Second Symposium on Solid Modeling and Applications, May 19-21, Montreal, Canada. Vol. 1, pp. 469-470. 1993.
- (Kim96) Kim, C. and P. J. O'Grady. *"A Representation Formalism for Feature-based Design"*. Computer Aided Design. Vol. 28(6/7), pp. 451-460. 1996.
- (Kiryama91) Kiriyama, T., T. Tomiyama, and H. Yoshikawa. *"The Use of Qualitative Physics for Integrated Design Object Modeling"*. (ASME-DE) DTM'91: Design Theory and Methodology, Miami, Florida, USA. September 22-25. Vol. 31, pp. 53-60. 1991.
- (Kraker97) Kraker, K. J. d. *"Feature Validation and Conversion"*. In *CAD Systems Development: Tools and Methods (ISBN: 3-540-62535-6)*, D. Roller and P. Brunet, Editors. Springer-Verlag. NY, USA, pp. 121-142. 1997.
- (Krause93) Krause, F.-L., H. Jansen, M. Biernert and F Major *"Product Modelling"*. Annals of the CIRP. Vol. 42/2, pp. 695-706. 1993.
- (Kumara94) Kumara, S. R. T., C.-Y. Kao, M. G. Gallagher and R. Kasturi. *"3-D Interacting Manufacturing Feature Recognition"*. Annals of the CIRP. Vol. 43/1, pp. 133-136. 1994.
- (Laakko93) Laakko, T. and M. Mäntylä. *"Feature Modelling by Incremental Feature Recognition"*. Computer Aided Design. Vol. 25(8), pp. 479-492. 1993.
- (Lee94) Lee, R.J.V., A.H.S. Al-Ashaab, and R.I.M. Young. *"Resolving Feature Interactions in Design for Injection Moulding"*. Proceedings of the 10th National Conference on Manufacturing Technology (NCMR'94) in Advances in Manufacturing Technology VIII, University of York. Vol. 10, pp. 274-278. 1994.

- (Lenau93) Lenau, T. and L. Mu. "*Features in Integrated Modelling of Products and their Production*". International Journal of Computer Integrated Manufacturing. Vol. 6(1-2), pp. 65-73. 1993.
- (Li90) Li, R.-K. and M.-H. Yu. "*A Framework for Prismatic Part-data Generation- Unit-Machined Loop Concept*". International Journal of Computer Integrated Manufacturing. Vol. 3(2), pp. 96-111. 1990.
- (Libardi86) Libardi Jr., E. C., J. R. Dixon, and M. K. Simmons. "*Designing with Features: Design and Analysis of Extrusions as an Example*". (ASME) Spring Nat. Design Engineering Conference and Show, Chicago, Illinois, USA. Vol. 86-DE-4, pp. 1-8. 1986.
- (Lim95) Lim, S. S., I. B. H. Lee, C. E. N. Lim and B. K. A. Ngoi. "*Computer-aided Concurrent Design of Product and Assembly Process: a Literature Review*". Journal of Design and Manufacturing. Vol. 5(2), pp. 67-88. 1995.
- (Luby86) Luby, S. C., J. R. Dixon, and M. K. Simmons. "*Creating and Using a Feature Data Base*". Journal of Mechanical Engineering. Vol. 5(3), pp. 25-33. 1986.
- (Mäntylä89) Mäntylä, M., J. Opas, and J. Puhakka. "*Generative Process Planning of Prismatic Parts by Feature Relaxation*". (ASME-DE) 15th Design Automation Conference, Montreal, Quebec, CA. Sept 17-21, 1989. Vol. 19-1, pp. 49-60. 1989.
- (Mäntylä94) Mäntylä, M., K. Lagus, T. Laakko and G. Sohlenius. "*Application of Constraint Propagation in Part Family Modelling*". Annals of the CIRP. Vol. 43/1, pp. 129-132. 1994.
- (Mäntylä96) Mäntylä, M., D. Nau, and J. Shah. "*Challenges in Feature-based Manufacturing Research*". Communications of the ACM. Vol. 39(2), pp. 77-85. 1996.

- (Mäntylä97) Mantylä, M. "Extracting Reusable Product Data". In *CAD Systems Development: Tools and Methods (ISBN: 3-540-62535-6)*, D. Roller and P. Brunet, Editor. Springer-Verlag. NY, USA, pp. 89-105. 1997.
- (Marefat93a) Marefat, M., S. Malhotra, and R. L. Kashyap. "Object-oriented Intelligent Computer-Integrated Design, Process Planning and Inspection" *IEEE Computer*. Vol. 26(3), pp. 54-65. 1993.
- (Marefat93b) Marefat, M., P. Banerjee, R. L. Kashyap and C. L. Moodie. "Capturing Intelligence as a Reusable Framework for Manufacturing Decision Processes". *International Journal of Production Research*. Vol. 31(8), pp. 1767-1795. 1993.
- (Marghitsu93) Marghitsu, D., A. H. Dogru, and D. B. Johnson. "Intelligent CAD Systems: A Requirement Study". (ASME-PD) *Computer Applications and Design Abstraction*. Vol. 49, pp. 151-156. 1993.
- (Martino94a) Martino, T. D. and F. Giannini. "The Role of Feature Recognition in Future CAD Systems". *IFIP International Conference on Feature Modelling and Recognition in Advanced CAD/CAM Systems, Valenciennes, FR*. Vol. 1, pp. 343-355. 1994.
- (Martino94b) Martino, T. D., B. Falcidieno, F. Giannini, S. Hassinger and J. Ovtcharova. "Feature-based Modelling by Integrating Design and Recognition Approaches". *Computer Aided Design*. Vol. 26(8), pp. 646-653. 1994.
- (Mayer94) Mayer, R. J., C. J. Su, T.-L. Sun. and R. A. Wysk "ECTOF: a Feature Representation Technique for Concurrent Engineering Applications". *Journal of Design and Manufacturing*. Vol. 4, pp. 49-65. 1994.
- (Medland93) Medland, A. J. and G. Mullineux. "A Constraint Approach to Feature-based Design". *International Journal of Computer Integrated Manufacturing*. Vol. 6(1-2), pp. 34-38. 1993.

- (Mill93) Mill, F. G., J. C. Salmon, and A. G. Pedley. "*Representation Problems in Feature-based Approaches to Design and Process Planning*". International Journal of Computer Integrated Manufacturing. Vol. 6(1-2), pp. 27-33. 1993.
- (Mitchell96) Mitchell, S. R. "*A Feature-based Approach to the Computer Aided Design of Sculptured Products*". PhD Thesis, Loughborough University, July 1996.
- (Molloy93) Molloy, E., H. Yang, and J. Browne. "*Feature-based Modelling in Design for Assembly*". International Journal of Computer Integrated Manufacturing. Vol. 6(1-2), pp. 112-125. 1993.
- (Murray93) Murray, J. L. and Y. Yue. "*Automatic Machining of 2,5D Components with ACIS Modeller*". International Journal of Computer Integrated Manufacturing. Vol. 6(1-2), pp. 94-104. 1993.
- (Nielsen91) Nielsen, E. H., J. R. Dixon, and E. E. Zinsmeister. "*Capturing and Using Designer Intent in a Design-with-Features System*". DTM'91: 3rd. International Conference on Design Theory and Methodology (ASME-DE), Miami, Florida, USA. Sept 22-25. Vol. 31, pp. 95-102. 1991.
- (Nnaji93) Nnaji, B. O., H.-C. Liu, and U. Rembold. "*A Product Modeller for Discrete Components*". International Journal of Production Research. Vol. 31(9), pp. 2017-2044. 1993.
- (Ohsuda89) Ohsuda, S. "*Toward intelligent CAD systems*". Computer Aided Design. Vol. 21(5), pp. 315-337. 1989.
- (Ovtcharova92) Ovtcharova, J., G. Pahl, and J. Rix. "*A Proposal for Feature Classification in Feature-based Design*". Computers and Graphics (Pergamon Press). Vol. 16(2), pp. 187-195. 1992.
- (Ovtcharova94) Ovtcharova, J. and U. Jasnoch. "*An Integration of Feature-Based Design and Consistency Management in CAD Management*". IFIP

International Conference on Feature Modeling and Recognition in Advanced CAD/CAM Systems, Valenciennes, May. Vol. 2, pp. 739-756. 1994.

(Patterson90) Patterson, D. W. "*Introduction to Artificial Intelligence and Expert Systems*". New-Jersey. Prentice-Hall Inc. Pub. 1990.

(Perng90) Perng, D.-B., Z. Chen, and R.-K. Li. "*Automatic 3D Machining Feature Extraction from 3D CSG Solid Input*". Computer Aided Design. Vol. 22(5), pp. 285-295. 1990.

(Perng97a) Perng, D.-B. and C.-F. Chan. "*A New Feature-based Design System with Dynamic Editing*". Computers in Industrial Engineering. Vol. 32(2), pp. 383-397. 1997.

(Perng97b) Perng, D.-B. and C.-F. Chang. "*Resolving Feature Interactions in 3D Part Editing*". Computer Aided Design. Vol. 29(10), pp. 687-699. 1997.

(Pratt85) Pratt, M. J. and P. R. Wilson. "*Requirements for Support of Form Features in a Solid Modelling System*". Final Report R-85-ASPP-01. CAM-I Inc., Arlington, Texas, USA. 1985.

(Pratt88) Pratt, M. J. "*Synthesis of an Optimal Approach to Form Feature Modelling*". (ASME) International Computers in Engineering Conference & Exhibition, California, USA. Vol. 1, pp. 263-274. 1988.

(Pratt93) Pratt, M. J. "*Application of Features Recognition in the Product Life-Cycle*". International Journal of Computer Integrated Manufacturing. Vol. 6(1-2), pp. 13-19. 1993.

(Regli96) Regli, W. C. and M. J. Pratt. "*What Are Feature Interactions ?*". (ASME) Design Engineering Technical Conference and International Computers in Engineering Conference, Irvine, California, USA. Vol. DFM-1285, pp. 1-12. 1996.

- (Rembold93)** Rembold, U., B. O. Nnaji, and A. Storr. "*Computer Integrated Manufacturing and Engineering*". Addison-Wesley Publishing Co. 1993.
- (Requicha80)** Requicha, A. A. G. "*Representations for Rigid Solids: Theory, Methods, and Systems*". ACM Computing Surveys. Vol. 12(4), pp. 437-464. 1980.
- (Requicha89a)** Requicha, A. A. G. "*Solid Modeling and Its Applications: Progress in Tolerancing, Inspection, and Feature Recognition*". 16th NSF Design & Manufacturing Systems Grantees Conference. Vol. 1, pp. 1-12. 1989.
- (Requicha89b)** Requicha, A. A. G. and J. H. Vanderbrande. "*Form Features for Mechanical Design and Manufacturing*". (ASME) International Computers in Engineering Conference and Exhibition. Vol. 1, pp. 47-52. 1989.
- (Requicha92)** Requicha, A. A. G. and J. R. Rossignac. "*Solid Modelling and Beyond*". IEEE Computer Graphics and Applications. Vol. 12(September), pp. 31-44. 1992.
- (Rimscha90)** Rimscha, M. v. "*Feature Modelling and Assembly Modelling - A Unified Approach*". IFIP/GI, WG 5. 2, Advanced Geometric Modelling for Engineering Applications, F.-L. Krause and H. Jansen Editors. Berlin, Germany. Elsevier Science Publishers B. V. (North-Holland). Vol. 1, pp. 203-214. 1990.
- (Rosen93)** Rosen, D. W. "*Feature-Based Design: Four Hypothesis for Future CAD Systems*". Research in Engineering Design. Vol. 5, pp. 125-139. 1993.
- (Rossignac90)** Rossignac, J. R. "*Issues on Feature-based Editing and Interrogation of Solid Models*". Computers and Graphics. Vol. 14(2), pp. 149-172. 1990.

- (Rossignac91) Rossignac, J. R. *"Through the Cracks of the Solid Modelling Milestone"*. Eurographics'91 Technical Report Series, Vienna- Austria- 2-6 Sept.. Vol. EG91STAR, pp. 23-109. 1991.
- (Salmon97) Salmon, J.C. *"Geometric Reasoning for Process Planning"*. PhD Thesis, The University of Edinburgh. December 1997.
- (Salomons93) Salomons, O. W., F. J. A. M. van Houten, and H. J. J. Kals. *"Review of Research in Feature-Based Design"*. Journal of Manufacturing Systems. Vol. 12(2), pp. 113-132. 1993.
- (Serrano88) Serrano, D. and D. Gossard. *"Constraint Management in MCAE"*. In *Artificial Intelligence in Engineering: Design*, J. S. Gero, Editor. Elsevier Science Publishers B. V. (North-Holland). Amsterdam. pp. 217-240. 1988.
- (Shah88a) Shah, J. J. and M. T. Rogers. *"Functional Requirements and Conceptual Design of Feature-based Modelling System"*. Computer-Aided Engineering Journal. Vol. 5(1), pp. 9-15. 1988.
- (Shah88b) Shah, J. J. *"Feature Transformation between Application-specific Feature Spaces"*. Computer-Aided Engineering Journal. Vol. 5(6-December), pp. 247-255. 1988.
- (Shah88c) Shah, J. J., P. C. Sreevalsan, M. T. Rogers, R. Billo and A. Mathew. *"Current Status of Feature Technology"*. Report R-88-GM-04.1 CAM-I Inc. Arlington, Texas, USA. 1988.
- (Shah88d) Shah, J. J. and M. T. Rogers. *"Feature Based Modeling Shell: Design and Implementation"*. (ASME) International Computers in Engineering Conference and Exhibition. Vol. 1, pp. 255-261. 1988.
- (Shah88e) Shah, J. J. and M. T. Rogers. *"Expert Form Feature Modelling Shell"*. Computer Aided Design. Vol. 20(9), pp. 515-524. 1988.

- (Shah90) Shah, J. J. "*Philosophical Development of Form Feature Concept*". Report P-90-PM-02. CAM-I Inc. Arlington, Texas, USA. Vol. 1990.
- (Shah91) Shah, J. J. "*Assessment of Feature Technology*". Computer Aided Design. Vol. 23(5), pp. 331-343. 1991.
- (Shah94a) Shah, J.J., et al. "*Comparative Study of Procedural and Declarative Feature Based Geometric Modelling*". IFIP International Conference on Feature Modeling and Recognition in Advanced CAD/CAM Systems, Valenciennes, May. Vol. 2, pp. 647-671. 1994.
- (Shah94b) Shah, J. J., A. Ali, and M. T. Rogers. "*Investigation of Declarative Feature Modeling*". (ASME) Computers in Engineering Conference and Exhibition, Minneapolis, Minnesota, USA, September, 11-14. Vol. 1, pp. 1-11. 1994.
- (Shah95) Shah, J. J. and M. Mäntylä. "*Parametric and Feature-Based CAD/CAM: Concepts, Techniques and Applications*". John Wiley and Sons Inc. 1995.
- (Sheu93) Sheu, L.-C. and J. T. Tin. "*Representation Scheme for Defining and Operating Form Features*". Computer Aided Design. Vol. 25(6), pp. 333-347. 1993.
- (Silva90) Silva, R. E. d., K. L. Wood, and J. J. Beaman. "*Representing and Manipulating Interacting Interfeature Relationships in Engineering Design for Manufacture*". (ASME) Advances in Design Automation. Vol. DE-23-1, pp. 1-8. 1990.
- (Sodhi91) Sodhi, R. and J. U. Turner. "*Representing Tolerance and Assembly Information in a Feature-Based Design Environment*". (ASME) Advances in Design Automation Conference. Vol. DE-32-1, pp. 101-108. 1991.
- (Sreevalsan92) Sreevalsan, P. C. and J. J. Shah. "*Unification of Form Feature Definition Methods*". IFIP Transaction B, WG 5. 2, Intelligent Computer

Aided Design, North-Holland, Elsevier Science Publishers. Vol. B-4, pp. 83-106. 1992.

(Srikantappa94) Srikantappa, A. B. and R. H. Crawford. "*Automatic Part Coding Based on Interfeature Relationship*". Manufacturing Research and Technology, Elsevier Science Publishers B. V. (North-Holland). Vol. 20, pp. 215-237. 1994.

(Stroud93) Stroud, I. "*Modelling Techniques for Handling Non-Geometric Information*". Second Symposium on Solid Modeling and Applications, May 19-21, Montreal, Canada. Vol. 1, pp. 367-376. 1993.

(Su94) Su, C. J., R. J. Mayer, T.-L. Sun and R. A. Wysk. "*A Three-phase Method for Feature Interaction Resolution*". Journal of Design and Manufacturing. Vol. 4, pp. 153-166. 1994.

(Subrahmanyam95) Subrahmanyam, S., W. DeVries, and M. J. Pratt. "*Feature Attributes and Their Role in Product Modeling*". Symposium on Solid Modeling and Applications. Vol. 1, pp. 115-124. 1995.

(Suh95a) Suh, H., R. S. Ahluwalia, and J. E. Miller. "*Dynamic Feature Generation During Design*". Journal of Design and Manufacturing. Vol. (5), pp. 115-126. 1995.

(Suh95b) Suh, H. and R. S. Ahluwalia. "*Feature Modification in Incremental Feature Generation*". Computer Aided Design. Vol. 27(8), pp. 627-635. 1995.

(Suzuki90) Suzuki, H., H. Ando, and F. Kimura. "*Geometric Constraints and Reasoning for Geometrical CAD Systems*". Computers and Graphics (Pergamon Press). Vol. 14(2), pp. 211-224. 1990.

(Talwar94) Talwar, R. and S. Manoochehri. "*Algorithms to Detect Geometric Interactions in a Feature-based Design System*". Advances in Design Automation (ASME-DE). Vol. 69-1(1), pp. 307-314. 1994.

- (Taylor96) Taylor, L. E. and M. R. Henderson. "*Validating a Feature-based Meta-model for Mechanical Products: a Case Study*". In *Advanced CAD/CAM Systems: State-of-the-art and future trends in feature technology*, R. Soenen and G. L. Olling, Editor. Chapman & Hall (IFIP). London. pp. 219-239. 1996.
- (Tomiya90) Tomiyama, T. and P. J. W. ten Hagen. "*Representing Knowledge in two Distinct Description: Extensional vs. Intensional*". *Artificial Intelligence in Engineering*. Vol. 5(1), pp. 23-32. 1990.
- (Tomiya93) Tomiyama, T., Y. Umeda, and H. Yoshikawa. "*A CAD for Functional Design*". *Annals of the CIRP*. Vol. 42/1, pp. 143-146. 1993.
- (Tseng94) Tseng, Y.-J. and S. B. Joshi. "*Recognizing Multiple Interpretations of Interacting Machining Features*". *Computer Aided Design*. Vol. 26(9), pp. 667-688. 1994.
- (Vancza93) Vancza, J. and A. Markus. "*Features and the Principle of Locality in Process Planning*". *International Journal of Computer Integrated Manufacturing*. Vol. 6(1-2), pp. 126-136. 1993.
- (Waco94) Waco, D. L. and Y. S. Kim. "*Geometric Reasoning for Machining Features using Convex Decomposition*". *Computer Aided Design*. Vol. 26(6), pp. 477-489. 1994.
- (Yoshikawa87) Yoshikawa, H. and K. Ando. "*Intelligent CAD in Manufacturing*". *Annals of the CIRP*. Vol. 36/1, pp. 77-80. 1987.
- (Young93) Young, R. I. M. and R. Bell. "*Design by Feature: Advantages and Limitations in Machining Planning Integration*". *International Journal of Computer Integrated Manufacturing*. Vol. 6(1-2), pp. 105-112. 1993.
- (Zeid91) Zeid, I. "*CAD/CAM: Theory and Practice*". *Computer Science Series*, McGraw Hill International Editions. 1991.

(Zhang93) Zhang, K. F. and H. A. ElMaraghy. "*Validity Check for a Functional-oriented Modeler*". (ASME) Advances in Design Automation (conf. code 19400, ISBN: 0-79-981181-6). Vol. **DE-65-2**, pp. 293-300. 1993.

(Zhang94) Zhang, Y. F., A. Y. C. Nee, and J. Y. H. Fuh. "*A Hybrid Approach to Computer-Aided Process Planning for Prismatic Parts*". (ASME) International Computers in Engineering Conference and Exhibition, Minneapolis, Minnesota, USA. September 11-14. Vol. **1**, pp. 437-444. 1994.

