



Analyse de modèles géométriques d'assemblages pour les structures et les enrichir avec des informations fonctionnelles

Ahmad Shahwan

► To cite this version:

Ahmad Shahwan. Analyse de modèles géométriques d'assemblages pour les structures et les enrichir avec des informations fonctionnelles. Autre [cs.OH]. Université de Grenoble, 2014. Français. NNT : 2014GRENM023 . tel-01071650

HAL Id: tel-01071650

<https://theses.hal.science/tel-01071650>

Submitted on 6 Oct 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Mathématiques et Informatique**

Arrêté ministériel : 7 août 2006

Présentée par

Ahmad SHAHWAN

Thèse dirigée par **Jean-Claude LÉON**
et codirigée par **Gilles Foucault**

préparée au sein **G-SCOP, Grenoble-INP**
et de **MSTII**

Processing Geometric Models of Assemblies to Structure and Enrich them with Functional Information

Thèse soutenue publiquement le **29 août 2014**,
devant le jury composé de :

M., John GERO

Professor, University of North Carolina, USA, Rapporteur

M., Marc DANIEL

Professeur, Université Aix-Marseille, Rapporteur

Mme, Marie-Christine ROUSSET

Professeur, Université Joseph Fourier, Examineur

M., Jean-Philippe PERNOT

Professeur, Arts et Métiers ParisTech, Examineur

M., Jean-Claude LEON

Professeur, Grenoble-INP, Directeur de thèse

M., Gilles FOUCAULT

Maître de Conférences, Université Joseph Fourier, Co-Directeur de thèse



Processing Geometric Models of Assemblies
to Structure and Enrich them with
Functional Information

Traitement de modèles géométriques
d'assemblages afin de les structurer et de les
enrichir avec des informations fonctionnelles

abstract

The digital mock-up (DMU) of a product has taken a central position in the product development process (PDP). It provides the geometric reference of the product assembly, as it defines the shape of each individual component, as well as the way components are put together. However, observations show that this geometric model is no more than a conventional representation of what the real product is. Additionally, and because of its pivotal role, the DMU is more and more required to provide information beyond mere geometry to be used in different stages of the PDP. An increasingly urging demand is functional information at different levels of the geometric representation of the assembly. This information is shown to be essential in phases such as geometric pre-processing for finite element analysis (FEA) purposes. In this work, an automated method is put forward that enriches a geometric model, which is the product DMU, with function information needed for FEA preparations. To this end, the initial geometry is restructured at different levels according to functional annotation needs. Prevailing industrial practices and representation conventions are taken into account in order to functionally interpret the pure geometric model that provides a starting point to the proposed method.

résumé

La maquette numérique d'un produit occupe une position centrale dans le processus de développement de produits. Elle est utilisée comme représentations de référence des produits, en définissant la forme géométrique de chaque composant, ainsi que les représentations simplifiées des liaisons entre composants. Toutefois, les observations montrent que ce modèle géométrique n'est qu'une représentation simplifiée du produit réel. De plus, et grâce à son rôle clé, la maquette numérique est de plus en plus utilisée pour structurer les informations non-géométriques qui sont ensuite utilisées dans diverses étapes du processus de développement de produits. Une exigence importante est d'accéder aux informations fonctionnelles à différents niveaux de la représentations géométrique d'un assemblage. Ces informations fonctionnelles s'avèrent essentielles pour préparer des analyses éléments finis. Dans ce travail, nous proposons une méthode automatisée afin d'enrichir le modèle géométrique extrait d'une maquette numérique avec les informations fonctionnelles nécessaires pour la préparation d'un modèle de simulation par éléments finis. Les pratiques industrielles et les représentations géométriques simplifiées sont prises en compte lors de l'interprétation d'un modèle purement géométrique qui constitue le point de départ de la méthode proposée.

Scientific Communications

Accepted

- SHAHWAN, A., LÉON, J.-C., FOUCAULT, G., AND FINE, L. Functional restructuring of CAD models for FEA purposes. *Engineering Computations* (2014).

Articles

- BOUSSUGE, F., SHAHWAN, A., LÉON, J.-C., HAHMANN, S., FOUCAULT, G., AND FINE, L. Template-based geometric transformations of a functionally enriched DMU into FE assembly models. *Computer-Aided Design and Applications* 11, 04 (2014), 436–449.
- SHAHWAN, A., LÉON, J.-C., FOUCAULT, G., TRLIN, M., AND PALOMBI, O. Qualitative behavioral reasoning from components' interfaces to components' functions for DMU adaption to fe analyses. *Computer-Aided Design* 45, 2 (2013), 383–394.

In proceedings

- SHAHWAN, A., FOUCAULT, G., LÉON, J.-C., AND FINE, L. Deriving functional properties of components from the analysis of digital mock-ups. In *Tools and Methods of Competitive Engineering* (Karlsruhe, Germany, 2012).
- LI, K., SHAHWAN, A., TRLIN, M., FOUCAULT, G., AND LÉON, J.-C. Automated contextual annotation of B-Rep CAD mechanical components deriving technology and symmetry information to support partial retrieval. In *Eurographics Workshop on 3D Object Retrieval* (Cagliari, Italy, 2012), pp. 67–70.

Contents

Acronyms	xiii
Introduction	xv
1 DMU and Polymorphic Representation	1
1.1 Introduction	1
1.2 Product model	2
1.3 Product prototype	4
1.4 Product digital mock-up	7
1.4.1 Computerized product models	7
1.4.2 DMUs as models and prototypes at a time	8
1.5 Geometric models and modeling methods	9
1.5.1 Geometric validity and the quality of a DMU	10
1.5.2 Discrete geometric models	13
1.5.3 Analytical geometric models	16
1.6 DMU as an assembly	18
1.6.1 DMU structure	19
1.6.2 Components' positioning	20
1.7 Other information associated to a DMU	25
1.8 Application of DMU	27
1.9 Basic principles of finite element analyses	28
1.9.1 Numerical approximations of physical phenomena	28
1.9.2 Generation of a FEM	30
1.10 DMU as polymorphic representation of a product	30
1.11 Adapted definition of DMU	31
1.12 Conclusions	32
2 Literature Overview	35
2.1 Introduction	35
2.2 Function formalization	36
2.3 Connections between form, behavior, and function	38
2.3.1 Behavior to complete the design puzzle	38
2.3.2 Pairs of interacting interfaces	38

2.3.3	Tools and guidelines to support the design process . . .	39
2.4	Constructive approaches to deduce function	41
2.4.1	Form feature recognition	42
2.4.2	Functionality as a result of geometric interactions . . .	44
2.5	Geometric analysis to detect interactions	46
2.5.1	Geometric interaction detection	46
2.5.2	Importance of a unique geometric representation . . .	47
2.6	CAD and knowledge representation	48
2.6.1	Domain knowledge and model knowledge	49
2.6.2	Ontologies as an assembly knowledge storehouse . . .	49
2.6.3	Knowledge-based engineering approaches	53
2.7	From CAD to FEA	55
2.7.1	Pre-processing at the core of the FEM	55
2.7.2	Direct geometric approaches	55
2.8	Conclusions	58
3	Functional Semantics: Needs and Objectives	61
3.1	Taking 3D models beyond manufacturing purposes	61
3.2	Differences between digital and real shapes	63
3.3	Enabling semi-automatic pre-processing	65
3.3.1	Pre-processing tasks	65
3.3.2	Pre-processing automation requirements	66
3.4	Bridging the gap with functional knowledge	67
3.5	Conclusion	70
4	Functional Restructuring and Annotation	73
4.1	Qualitative bottom-up approach	74
4.2	Common concepts	74
4.2.1	Function as the semantics of design	75
4.2.2	Functional Interface	76
4.2.3	Functional Designation	78
4.2.4	Functional Cluster	81
4.2.5	Conventional Interface	85
4.2.6	Taxonomies	87
4.3	Method walk-through	91
4.4	Conclusions	93
5	Functional Geometric Interaction	95
5.1	Functional surfaces	95
5.2	Geometric preparation and rapid detection of interactions . .	97
5.2.1	Geometric model as global input	97
5.2.2	Maximal edges and surfaces	98
5.2.3	Geometric interaction detection	99
5.2.4	Local coordinate systems	103

5.2.5	Conventional interface graph	104
5.3	Precise detection of interaction zones	107
5.4	Form-functionality mapping	108
5.4.1	Multiple functional interpretation	109
5.5	Conclusions	110
6	Qualitative Behavioral Analysis	111
6.1	Behavioral study to bind form to functionality	112
6.2	Reference states	112
6.3	Qualitative representation of physical properties	114
6.3.1	Qualitative physical dimension	115
6.3.2	Algebraic structure of qualitative values	121
6.3.3	Coordinate systems alignment	123
6.4	Reference state I: Static equilibrium	124
6.4.1	Static equilibrium equations	124
6.4.2	Graph search to eliminate irrelevant FIs	125
6.4.3	Local failure of functional interpretation	129
6.4.4	Graph search example	129
6.5	Reference state II: Static determinacy	130
6.5.1	Statically indeterminate configurations	133
6.5.2	Force propagation and force propagation graphs	137
6.6	Reference state III: Assembly joint with threaded link	138
6.6.1	Detection of force propagation cycles	139
6.7	Conclusions	142
7	Rule-based Reasoning	145
7.1	Knowledge at the functional unit level	146
7.2	Inference rules as domain knowledge	146
7.3	Reasoning alternatives	148
7.3.1	Dynamic formalization of domain specific rules	148
7.3.2	Problem decidability	149
7.4	DMU knowledge representation	150
7.4.1	Ontology definition through its concepts and roles	151
7.4.2	Ontology population with model knowledge	152
7.5	Formal reasoning to complete functional knowledge	154
7.5.1	Inference rules in DL	155
7.5.2	The unique name assumption	157
7.5.3	The open world assumption	158
7.5.4	Integration of DL reasoners	159
7.6	Conclusions	163

8	Results and Comparative Study	165
8.1	Application architecture	165
8.2	Application to industrial examples	167
8.3	Integration with FEA pre-processors	173
8.4	Conclusions	175
9	Conclusions and Perspectives	177
9.1	Conclusions	177
9.2	Perspectives	180
A	Fit Tolerancing and Dimensioning	183
B	Dual Vectors	187
C	Screw Theory	189
D	Description Logic	193

List of Definitions

1.1	Definition (Product model)	2
1.2	Definition (Product prototype)	4
1.3	Definition (Digital mock-up)	32
4.1	Definition (Function)	75
4.2	Definition (Functional Interface)	77
4.3	Definition (Functional Designation)	78
4.4	Definition (Functional Cluster)	81
4.5	Definition (Conventional Interface)	87
4.6	Definition (Taxonomy)	88
5.1	Definition (Conventional interface graph)	104
5.2	Definition (Functional interpretation)	109
6.1	Definition (Reference state)	112
6.2	Definition (Force Propagation Graph)	137
B.1	Definition (General dual number ring)	187
B.2	Definition (General dual number semi-ring)	188
C.1	Definition (Screw)	189
C.2	Definition (Reciprocal screws)	191

List of Hypotheses

5.1	Hypothesis (Functional surfaces)	96
6.1	Hypothesis (Rigid bodies)	113
6.2	Hypothesis (Conservative systems)	114
6.3	Hypothesis (Mechanical interactions)	114
6.4	Hypothesis (Static equilibrium)	124
6.5	Hypothesis (Static determinacy)	134
6.6	Hypothesis (Force propagation)	139

Acronyms

AI	Artificial Intelligence.
API	Application Programming Interface.
ARM	Assembly Relation Model.
B-Rep	Boundary representation.
BOM	Bill of materials.
C&CM	Contact and Channel Model.
CAD	Computer Aided Design.
CAM	Computer Aided Manufacturing.
CAPP	Computer Aided Process Planning.
CI	Conventional interface.
CIG	Conventional interface graph.
CIuG	Conventional interface underling undirected graph.
CNC	Computer Numeric Control.
CPM	Core Product Model.
CSG	Constructive Solid Geometry.
CWA	Closed World Assumption.
DIG	DL Implementation Group.
DL	Description Logic.
DMU	Digital mock-up.
DoF	Degree of freedom.
FBS	Function-Behavior-Structure.
FC	Functional cluster.
FD	Functional designation.
FE	Finite element.
FEA	Finite element analysis.
FEM	Finite Element Model.
FI	Functional interface.
FOL	First Order Logic.

FPG	Force propagation graph.
FR	Feature Recognition.
GARD	Generic Assembly Relationship Diagram.
GCI	General Concept Inclusion.
GD&T	Geometric Dimensioning and Tolerancing.
GPU	Graphics Processing Unit.
KBE	Knowledge Based Engineering.
KRR	Knowledge Representation and Reasoning.
MDB	Model-Based Definition.
NURBS	Non-Uniform Rational B-Spline.
OAM	Open Assembly Model.
OIL	Ontology Interchange Language.
OWA	Open World Assumption.
OWL	Web Ontology Language.
PDP	Product development process.
PFM	Part Function Model.
PLM	Product Lifecycle Management.
RDF	Resource Description Framework.
RDF-S	Resource Description Framework Scheme.
RS	Reference state.
SPARQL	SPARQL Protocol and RDF Query Language.
SSWAP	Simple Semantic Web Architecture and Protocol.
UNA	Unique Name Assumption.
URI	Uniform Resource Identifier.
VR	Virtual Reality.
W3C	World Wide Web Consortium.
XML	Extensible Markup Language.

Introduction

When designing an artifact that is identified by its functionality, it is a common practice to decompose the artifact in question into components, each satisfying a well-defined set of functions that, put together, lead to the satisfaction of the desired functionality of the designed artifact.

In the industrial context, components happen to be physical objects, defined by their shapes and materials that decide their physical properties and behaviors. In order for an object to deliver a precise function, its shape has to be carefully engineered. The 3D *shape* of the object dictates its interactions with its environment, i.e., its neighboring components, its neighboring products or a neighboring human being. These interactions define its *behavior*, thus its *functionality*.

Because of this pivotal importance of components shapes to deliver their functions, tools were provided and conventions established to enable the production and communication of shape design models as part of the product development process (PDP). This emphasis is a natural outcome of a shape-oriented design process. Design intentions, however, are not clearly reflected in design models, in spite of their clear presence in engineers' minds during the design process. In fact, no robust tools or agreed-upon conventions exist to link a particular design with its rationale.

This observation used to be less pronounced at the time blueprints were used to define design models. Blueprints are 2D drawings that aim at unambiguously defining the shape of an object. They have been in use for so long that conventions converged toward globally understood agreements, and standards were put to govern such conventions [16, 183]. Nevertheless, the advent of Computer Aided Design (CAD) systems in early 80s soon provided designers with another geometric dimension that would remarkably influence industrial standards and conventions. 3D solid modelers prevailed as a natural choice for product design, engineers shifted to producing 3D models instead of traditional technical drawings, and mechanical components became dominantly represented as 3D objects in today's models. This gave birth to the concept of digital mock-up (DMU), which gathered the representation of components of a product assembly in one geometric model.

Efforts were paid to centralize the product knowledge in one place, and

the DMU was suggested as a natural candidate as it geometrically defines the product. In spite of attempts to homogenize and standardize the representations of non-geometric knowledge [12], defined standards are still poorly implemented in industrial practices because commercial software products are far from exploiting these standards. In fact, an industrial DMU, as currently available, is no more than a conventional geometric representation of a product assembly. A DMU can at best contain loose textual annotations, which may be interpreted within an organization or a working group, if at all interpretable. This is partially because a textual annotation does not relate precisely to a geometric subset of a component or an assembly.

The need of design intentions, however, remained paramount, if the DMU is to be fully exploited in the PDP, and utilized beyond Computer Aided Manufacturing (CAM) applications. In fact, this knowledge is still being mined from geometric models of a product to feed applications such as geometric pre-processing of assembly models for simulation purposes. This is particularly the case of mechanical simulations where the structural behavior is a key issue that is commonly addressed using numerical methods such as the Finite Element Model (FEM). However, and due to conventional representations of functional and technological information in the DMU, the model preparation task for FEM is still mainly manual and resource intensive. This is particularly true for complex products like aircraft structures [1].

The user-intensive functional annotation of a DMU introduces a bottleneck into today's highly automated PDP. In order to accelerate product development, an automated method should be established that enables the extraction of relevant functional information out of pure geometric representation of product assembly. Function is a key concept for designers and engineers that closely relates to the design activity and, hence, to the so-called design intent [107]. Consequently, it is highly important to provide engineers and designers with this functional information tightly connected with 3D component models, so that they can efficiently process them during the PDP. Furthermore, the desired approach should take into account mainstream industrial practices and conventions when interpreting geometric models.

In this work, the focus is placed on the application to structural behavior of a product or, more precisely, of an assembly of components. The proposed method is an enrichment process that mainly aims at a seamless integration with geometric preprocessors for finite element (FE) simulation purposes, even though other applications can also be envisaged. In order for this method to provide an adequate input to finite element analysis (FEA) applications functional annotations and component denominations should be made in tight connection to precise geometric entities that they describe. To this end, geometry processing and reasoning mechanisms applied to mechanical behaviors are set up to adequately structure geometric models of

assemblies.

In the rest of this document, Chapter 1 provides an introductory presentation of industrial concepts that relate to our work. It particularly presents what can be expected from an industrial DMU nowadays. Literature and work related to the proposed method is reviewed and analyzed in Chapter 2. Chapter 3 sheds more light on the motivation of our work, and the role that the proposed method plays in an efficient PDP. Chapter 4 defines concepts and terminology that are used across this document and upon which the proposed method is founded. It also provides an overview of this method before later chapters develop further on each stage.

Chapter 5 develops in more details the geometric analysis of the input model, which is the pure geometric model of a DMU. This chapter shows how interactions between components are reconstituted on a geometric basis and how functional interpretations can be assigned to each of them. At this stage, the shape – function relationship cannot be unambiguously recovered.

Chapter 6 then provides the means to functionally interpret those interactions in an unambiguous way, through a qualitative behavioral analysis of the model. This is algorithmic approach achieved through the tight dependencies between shape, function and behavior that produce a unique relation between shape and function for the interactions between components. The concept of reference states is then used to synthesize some component behavior through their interactions in order to reject irrelevant configurations, thus removing ambiguities. Further qualitative behavioral information is derived too.

Chapter 7 completes the functional picture of the assembly using domain specific rules and taking the functional interpretation beyond the interaction level, toward the functional unit level, using the effective relationship between shape and function at the interface level and the newly derived behavioral information at the component and component cluster levels. It is an inference-based reasoning approach that can be adapted to the conventional representations of assemblies and meet the current practices observed in industry.

Once the input model is geometrically restructured, and functionally annotated, it is made available as input for FEA preprocessors. Chapter 8 shows results of the application of the proposed method on examples varying from illustrative models to industrial scale DMUs. The same chapter also shows how the method successfully lends hand to a template-based geometric preprocessor, generating simulation models that correspond to the simulation objectives. Chapter 9 concludes this document, exploring potentials of future work to extend the proposed method and its application.

Chapter 1

Digital Mock-Up and the Polymorphic Representation of Assemblies

DMUs constitute a starting point to our research. Thus, it is indispensable to present basic concepts and definitions that are central to this work before detailing our approach. Those concepts are presented from different viewpoints according to the literature and to industrial practices, before an adaption of these concepts to our context is underlined. An analysis of a DMU content also shows how it can refer not only to a single representation of an assembly but to a polymorphic one.

1.1 Introduction

In this chapter we provide a general understanding of a DMU, a concept which is central to our research. Then, we formally define this concept as it applies to the current work. To this end, we first present closely related notions that pave the way to the conceptualization of a DMU. We also show what kind of information it holds, and how this information is represented.

Sections 1.2 and 1.3 demonstrate and distinguish two concepts: the *product model* and *product prototypes*. Though these terminologies are used interchangeably across literature, we clearly make the distinction according to our understanding, and to the context of this work. Then, the concept of DMU is analyzed in the following sections to address its representation from a geometric point of view as well as from a more technological point of view through the concept of assembly. This leads to the analysis of the effective content of an assembly and its relationship with a DMU. Subsequently, the generation of Finite Element Analyses from a DMU is outlined to illustrate into which extent a DMU can contribute to define a Finite Element Model.

This finally leads to the concept of polymorphism of an assembly.

1.2 Product model

In the context of product development, manufacturing processes of this product must be precisely defined, so that the resulting product matches its initial requirements, i.e., the product meets designers' and users' expectations. To this end, models are used to define the product in enough details as the outcome of an unambiguous and overall manufacturing process.

Models consist of documents and schemes that describe the product, often visually. They often use common languages and annotations to refer to resources (materials, quantities, etc.) and processes (parameters, dimensions, units, etc.). Those annotations should be standardized, or at least agreed upon among people involved in the production process. Otherwise a model can be misinterpreted.

Definition 1.1 (Product model). A product model is a document, or a set of documents, that uniquely define the manufacturing process of a product in compliance with its specifications [43, 140].

In this sense, models can be viewed as cookbooks showing how to produce instances of the product that conform to the same specifications. Models are closely related the production process for the following reasons:

- *To persistently capture the know-how of the production process.* In the absence of such documentations, the manufacturing knowledge is only present in engineers' minds, making this process highly dependent on the availability of experts. Models capture this knowledge and reduce the risk associated with such dependency;
- *To formally define the manufacturing process, leaving no room for ambiguity and multiple interpretations.* This formality allows for the reproduction of identical instances of the same 'pattern', avoiding undesirable surprises due to miscommunication or improvisation of incomplete specifications. Otherwise, divergence in the final product may drift it away from initial requirements;
- *To allow tracking of and easy adaption to requirements.* A product (or its prototype, as shown in Section 1.3) still may fail to fulfill the desired requirement. In this case, the product should be re-engineered. The existence of a model allows engineers to perform more easily modifications that can be directly mapped to the product characteristics to be amended. Another case when the product, thus its model, is to be re-engineered is when the requirements evolve, which is likely to happen in almost all industries.

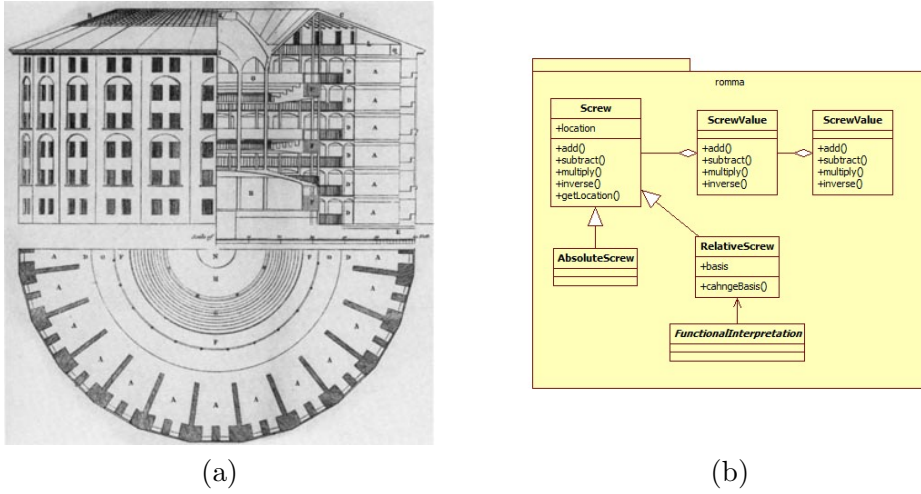


Figure 1.1: Examples of partial product model: (a) architectural blueprints; (b) software diagram.

Product models existed quite early in different engineering domains such as architecture, mechanics, electrics, electronics, and computer software, among others. These models were not digital until a couple of decades ago. Depending on the discipline, some subsets of these models can be referred to as technical drawings, blueprints, draftings, diagrams, etc. Figure 1.1 shows examples of such models in different disciplines and applications.













As pointed out earlier, the current concept of product model focuses essentially on manufacturing issues and does neither incorporate properties that ensure the consistency between its set of documents and the product obtained nor cover some parts of the product design process.

Product model in the field of mechanical engineering

When it comes to mechanical engineering, product models were traditionally referred to as *technical drawings* or *drafting*. In fact, those are 2D drawings that represents either a projection of the product onto a given plane according to a given orientation (usually perpendicular to the plane and aligned with a reference direction) and/or a cross section into the product components(s) [70]. These drawings form a part of the product model.

As Figure 1.2 shows, precise annotations are used to augment those drawings with complementary information such as Geometric Dimensioning and Tolerancing (GD&T), some of which cannot be geometrically represented on a sheet. Such information is mandatory to allow people manufacturing the product [124]. This figure shows in red, shaft/housing tolerancing and dimensioning symbols explained further in Appendix A. Projection, cross-

Table 1.1: Geometric tolerancing reference chart as per ASME Y14.5 – 1982.

	straightness		planarity		circularity
	cylindricity		line profile		surface profile
	perpendicularity		angularity		parallelism
	symmetry		position		concentricity

sectioning and annotations follow agreed-upon conventions that make the model as unambiguous as possible to a knowledgeable reader. Table 1.1 show standard dimensioning and tolerancing symbols as defined by ASME Y14.5 – 1982 [183].

1.3 Product prototype

It is preferable that design defects be outlined as early as possible in the product live cycle. More specifically, it is of high advantage that a shortcoming be reported before a real instance of a product is manufactured and machined. This is due to the high cost of machining and other manufacturing processes. If compliance tests are to be run directly on the real product, without any previous test on some sort of a “dummy” version of it, a considerable risk is involved since the product is likely to be re-engineered. The manufacturing and machining costs can be nearly doubled at each iteration.

To this end, a prototype, close enough in its behaviors to the real product but with reduced production costs, is produced first. Then, different tests are run against this prototype to assess its conformity to different requirements and detect potential deficiencies. Whenever such shortcomings are revealed, the product model is adapted accordingly, generating a new prototype. Then, the process is repeated until the prototype is validated by all tests. This can be seen as an iterative process of modeling, prototyping, and evaluation. Subsequently, the product is progressively refined through multiple iterations.

Definition 1.2 (Product prototype). A product prototype is a *dummy* representation of a real product, that is meant to emulate it in one or more aspects. It is used to assess or predict certain behaviors and/or interactions [160, 184].

Prototypes can emulate the product functionally, aesthetically, physically, ergonomically, etc. depending on the intended assessment planned on the prototype. Prototypes are vital in an efficient production process for the following reasons:

- *To allow early recognition of deficiencies.* The earlier the deficiencies are detected, the lower the amendment cost is, since fewer stages are wasted and redone;

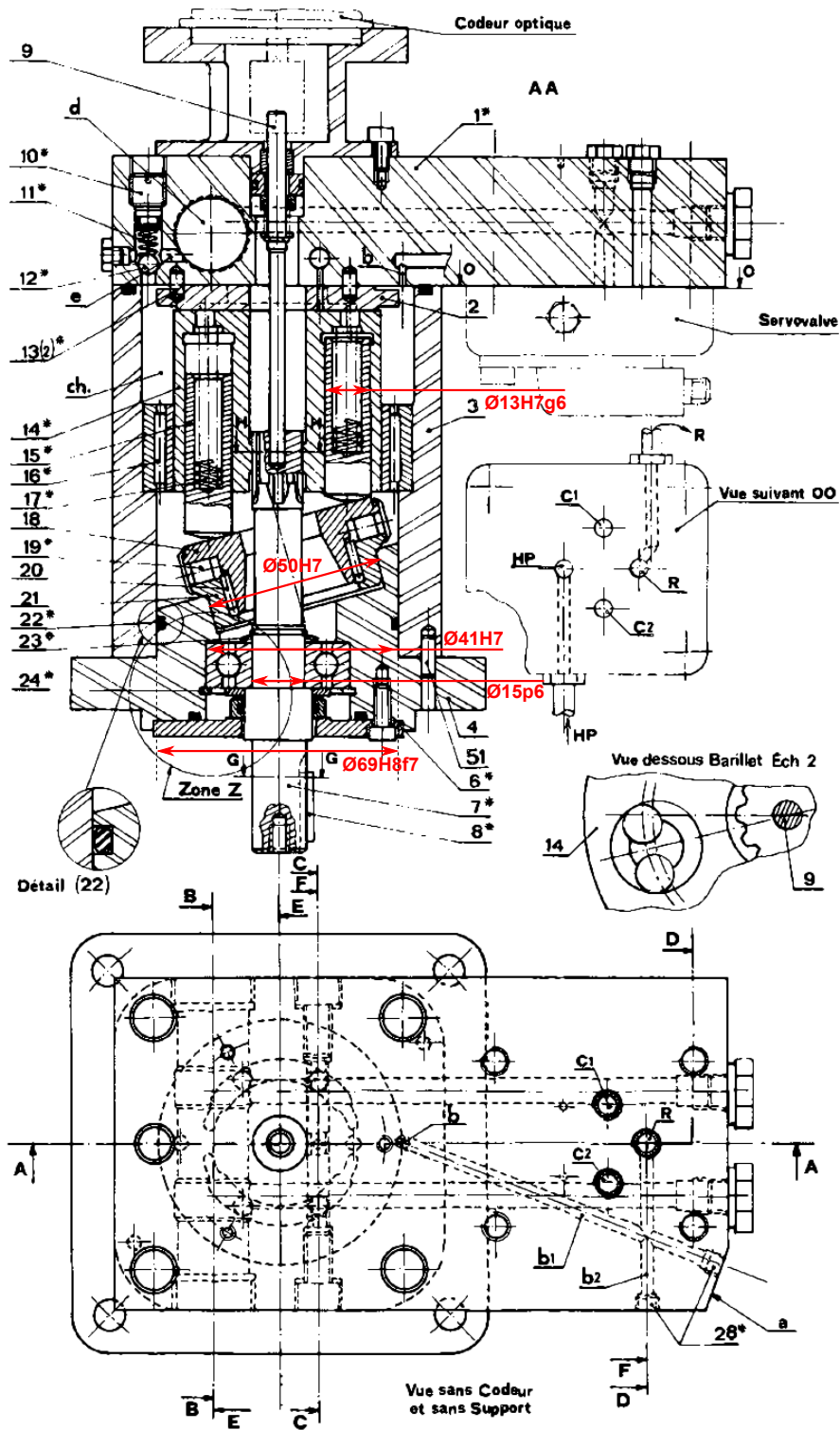


Figure 1.2: Blueprint of a mechanical product, showing a cross-section (top) and a projection (bottom). The projection also shows the cutting plane of the cross-section. The drawing shows in red, shaft/housing tolerance annotations.

- *To allow testing on life-critical products.* Some highly critical industries, such as aeronautics, tolerate little or no failure once the product is put to operation. Errors can be fatal at this stage. Thus, prototypes allowing virtual tests on the product are necessary in such cases;
- *To help decision making.* Studies show that decisions taken at early stages of a design process are highly expensive [47, 180]. However, oftentimes product behavior and the impact that it may entail cannot be precisely predicted at those stages. Prototypes enables engineers to do a sort of what-if analyses, and benefit from their feedbacks before taking a final decision about the product design.

More recently, product prototypes evolved toward digital or virtual ones, which reduces further a product development process and can be achieved using digital simulations.

It is worth noticing that tests run against prototypes do not replace quality and compliance tests that should be run on the real product. Product prototypes are just mock-ups, they emulate the products behavior to a certain extent, but not exactly.

Prototypes are used in different engineering disciplines. In architecture and interior design scaled-down prototypes are used to give a global perspective of the structure before it is actually implemented (see Figure 1.3a). Architecture prototypes are often used for aesthetic and ergonomic assessments.

In software engineering, incomplete versions of the software that fulfill certain requirements are implemented first to satisfy unit tests. Unit testing is in the core of software engineering best practices to avoid bulk debugging. Usually, one module of the software is tested at a time, with the rest replaced by mock modules.

Product prototype in the field of mechanical engineering

Approximate replicas of a mechanical product may be built to assess its ease of use, functionality, structural behavior, and so on. Those replicas are prototypes that are very similar to the designed product (see Figure 1.3a).

However, a product and its prototype differ in how and from what each is made. Materials of the real product are usually costly, thus prototypes are built out of cheaper materials that have similar physical properties according to the intended tests. Moreover, the manufacturing and machining processes of the real product are often expensive as well, partially due to the choice for materials. Then, prototypes are crafted using different methods that reduce costs, keeping the final shapes as close as possible to the original design [160].

Figure 1.4 shows a prototype of a hand navigator [49]. The prototype is made of thermoplastic powder shaped by means of selective heat sintering.



Figure 1.3: Examples of product prototypes: (a) Architectural scale prototype of the interior of a building; (b) Full-size car prototype.

Though the resulting object is perfectly fitted for concept proofing, machining techniques and materials are not suitable for mass production, once the product is approved.

Despite their minimized cost, prototypes are often wasteful and non-reusable (apart in some discipline, like software engineering, where prototypes can later be integrated in an operational product). This makes the manufacturing process redundant: one manufacturing process at least for prototyping, and then another one for real production. It would be highly advantageous if some test could be directly run against the models themselves, without the need to create a prototype.

1.4 Product digital mock-up

Sections 1.2 and 1.3 introduced product model and product prototype as historically two separate concepts. However, technological advances in information systems allowed engineers to merge those concepts into a single one, introducing little by little what became known as DMU in the domain of mechanical engineering.

1.4.1 Computerized product models

With the introduction of information technology and its applications, engineering and production disciplines tended to make the most out of its possibilities. One obvious application was modeling. Engineers and designers soon got convinced to use computers instead of drafting tables to materialize their designs. This gave way to CAD systems, who were based on advances in computer-based geometric modelers. Geometric modelers were first two-dimensional, and offered little advantage over classical draft-



Figure 1.4: Hand navigator prototype (Chardonnet & Léon [49]).

ing, apart their ease of use. Soon, those modelers started to address 3D solid modeling and integrated complementary facilities such as parametric and feature-based modeling [121, 54]. Digital product models became easier to produce and to interpret.

With the advent of Model-Based Definition (MDB) paradigm [140, 43], these models were soon imported into the downstream manufacturing process, to allow what is now called CAM. CAD models contained information not only understandable by expert engineers, but also by machine tools to automatically configure some of the product manufacturing processes.

An important revolution in the field of CAD was the introduction of 3D modelers, thanks to the fast-paced advancement of computer graphics. This gave the designers a better perception of their work, even though incomplete. 3D models now allow engineers to perform basic prototyping, at least from an aesthetic point of view, with categories of shapes as prescribed by CAD systems. Indeed, each CAD software enables the generation of component/product shape within a range prescribed by its algorithms. As a result, CAD software can detect some geometric inconsistencies, e.g., self-intersections, invalid topologies, interferences, etc. (see Section 1.5.1 for a discussion on geometric validity of a DMU), when a component shape/product falls outside the range of shapes it can describe.

Product models have become more than mere patterns that used to dictate how the product should be manufactured. The line that separated models from prototypes got thinner as more and more product assessments can be readily conducted on the models themselves.

1.4.2 Digital mock-ups as models and prototypes at a time

Computerized product models that also played the role of prototypes are commonly called digital mock-ups (DMUs). They mainly contain the 3D

geometric model of a product, but are not restricted to that. As product models they also incorporate supplementary information about material and other technological parameters.

The goal of DMUs is not limited to manufacturing only. Now that they provide detailed geometry alongside material physical properties, different physical simulations can be set up, taking advantage of increasing computational capabilities rather than generating physical prototypes.

DMUs can be seen as the result of advances in geometric modeling software and CAx systems. They directly support manufacturing processes, fulfilling the role of product referential models, as well as the basis of simulation mock-ups, and serving as product digital prototypes [177]. By the late 90s, a DMU was seen as a realistic computer simulation of a product, with the capability of all required functions from design engineering, manufacturing, product service, up to maintenance and product recycling [57]. From this perspective, the DMU stems from the merge of product model and product prototype.

Product geometry is a key information around which the DMU is organized. Figure 1.5 shows an example of a DMU of a centrifugal pump as visualized by its 3D representation. Other types of information, essential for manufacturing and prototyping purposes are also present in the DMU, and will be discussed in more details in Section 1.7.

In this sense, the DMU works as a repository of the engineering knowledge about a product that can be used throughout its life cycle [47]. Thus, DMUs are seen as the backbone of the product development process in today's industries [64].

Figure 1.6 shows how the generation of technical drawings can be partly automated using the 3D geometry of CAD models out of the product DMU. Then, GD&T can be carried out by engineers to add technological data.

1.5 Geometric models and modeling methods

Often CAD systems consider a DMU as a set of components, that may also be called parts, assembled together to directly form the 3D representation of a product, or to form modules (sub-assemblies) that in turn are assembled into a product. Section 1.6 explains different methods and viewpoints about component assembly. In this section we are more concerned about how a component is represented geometrically in a CAD system.

Geometric modelers are as important to CAD systems as the product geometric model is to DMUs. The geometric modeling process is highly influenced by the category of geometric model attached to a CAD system. Often, engineers choose to represent a component as a volume; a three-dimensional manifold [131] that divides the 3D-euclidean space into three sets: its interior C , its boundary ∂C , and its exterior $\sim C$. Then, the ge-

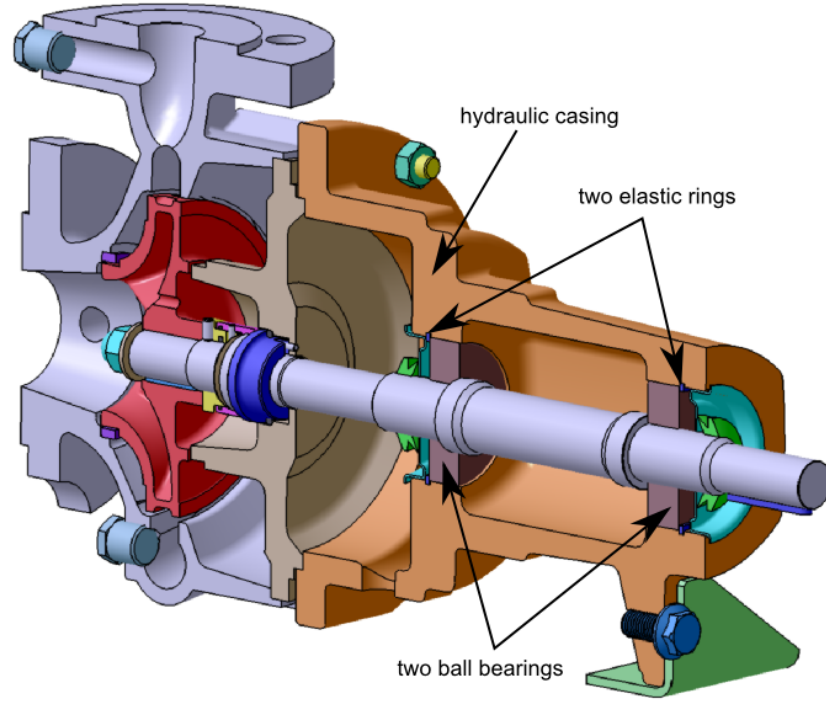


Figure 1.5: A DMU geometry of a centrifugal pump, showing different parts using colors. For a better understanding of the product shape, the DMU is sectioned by a vertical plane.

ometric model commonly used in CAD systems is of type boundary representation (B-Rep) [119, 120]. In this case, the material of a component is described by the topological closure of its interior $cl(C)$, which is the union of its interior and its boundary $cl(C) = C \cup \partial C$ [120].

1.5.1 Geometric validity and the quality of a DMU

As digital geometric representations of a product, a DMU may contain unrealistic, or unrealizable, configurations. An example configuration that is frequently encountered in industrial models is the volumetric interference between two solids. This configuration can lead to several interpretations:

- a. It might be a by-product of an imprecise design and it is therefore incorrect;
- b. It might also be a deliberate artifact to reflect some conventional meaning and, in this case, it has no impact on the correctness of the DMU.

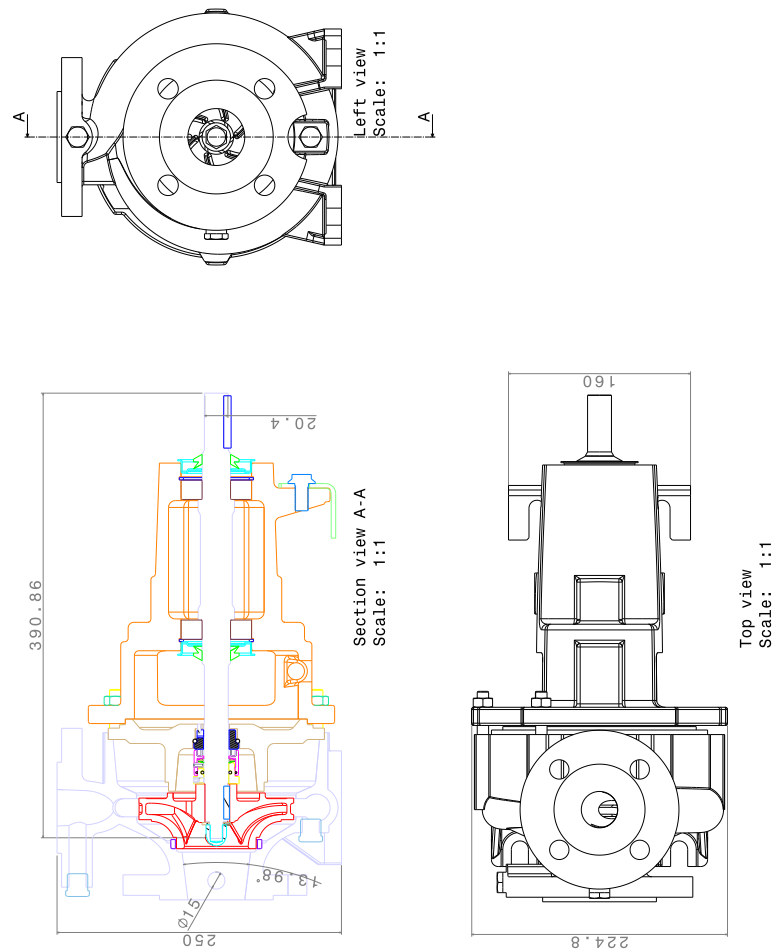


Figure 1.6: Automated generation of a technical drawing from a DMU that contribute to the definition of a product model.

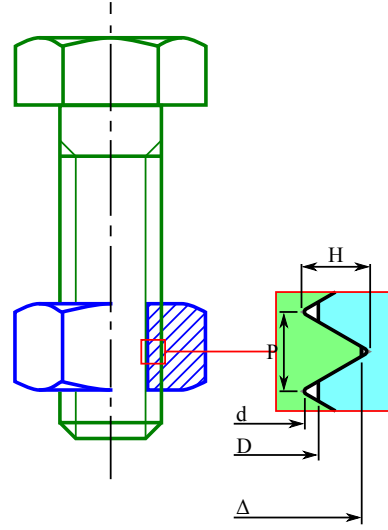


Figure 1.7: A cross section of a threaded link between a screw and a nut represented as a simple interference in a 3D assembly geometric model. The sub-figure at the right shows how the cross section may look like in a real product, and the technological parameters it may have conveyed. H : height of the thread; P : pitch; d : minor diameter of external thread; D : minor diameter of internal thread; Δ : nominal diameter.

Figure 1.7 shows an example where a threaded link is represented as such an interference, which falls into the interpretation of type b.

Furthermore, some geometric modelers let a user create non-manifold configurations (see Figure 1.8). Some of these configurations are useful to produce a simplified representation of the real object that is needed to perform mechanical simulations using the finite element method (see Figure 1.8c and Section 1.9). Those configurations, however, are not physically realizable [131].

These unrealistic or unrealizable geometric arrangements put a question about the quality of DMU. One may ask what geometry to consider as valid, and what to reject or disallow, knowing that these configurations cannot be filtered out by the algorithms of a geometric modeler. In fact, the answer to this question highly depends on conventions being followed by the users or engineers because there is no representation standard that is used in geometric modelers to discard such arrangements. However, studying industrial models showed that there exists a general consensus in the domain of mechanical engineering that geometric degeneracies such as non-manifold configurations (see Figure 1.8a, b) should be avoided in a DMU, as they are often misleading as for how to be interpreted. Meanwhile, volumetric intersections are largely accepted, as they convey a particular meaning.

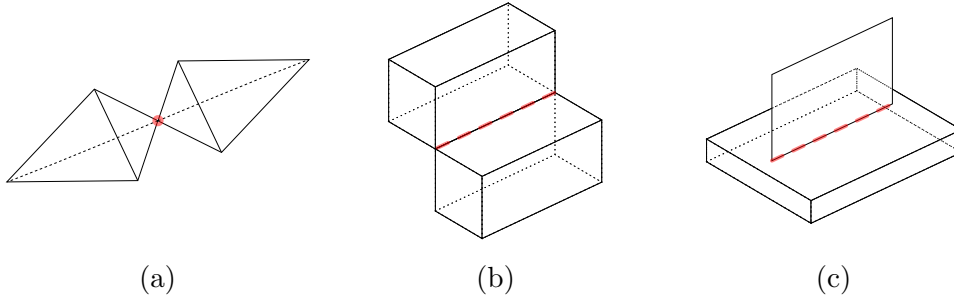


Figure 1.8: (a), (b), (c) Geometric models with highlighted non-manifold configurations. (c) is an example of simplified representation that can be used for a mechanical simulation.

Manifold or not, digital geometric models defining products in CAD systems have their boundary represented either with faceted models, i.e., piecewise linear surfaces, or with piecewise smooth surfaces. Here, the first category is named *discrete geometric models* and the second one *analytic geometric models*.

1.5.2 Discrete geometric models

Discrete geometric models consist of a finite set of geometric elements topologically connected to each other to define the boundary of a shape. These elements are manifolds that can be either one, two, or three-dimensional.

The very basic geometric element is a *vertex*: a point lying in 1D, 2D or 3D-space, this is a zero-dimensional manifold.

Two vertices connect to each other defining a line segment or *edge* that is a one-dimensional manifold. An aggregation of edges on the same plane can form a piecewise 1D curve. If every vertex of this aggregation is topologically connected with at most two edges per connection¹ the curve is indeed a one-dimensional manifold.

A 1D closed² manifold and planar curve with no self-intersection bounds a discrete planar area or *face*, i.e., a two-dimensional geometric entity. An aggregation of faces connected to each other forms a faceted 2D surface. If every edge of this aggregation is topologically connected with at most two faces, while the connection between faces happens uniquely at their boundary, i.e., at the edge level, the surface is a two-dimensional manifold.

A 2D closed³ manifold surface with no self-intersections bounds a *solid*,

¹Connection between edges happens at their boundary, i.e., either of their vertices.

²A closed curve is a curve in which a connection happens at every vertex of each of its edges.

³A closed surface is a surface in which a connection happens at every edge of each of

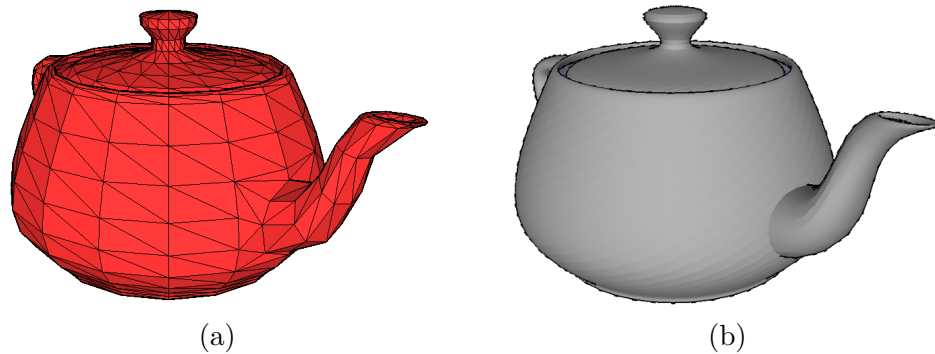


Figure 1.9: Geometric models of a teapot: (a) discrete model (triangular surface mesh); (b) analytical model (composite free-form shape obtained from a set of surface patches).

a three-dimensional geometric entity. Solids can be aggregated to form more complex ones. If this aggregation is topologically connected in a way that connection happens uniquely at face level the resulting solid is manifold.

Discretized models are also called *meshes*. Meshed objects used in a product development process to describe a solid are represented in one of two ways:

Surface meshes

Using a discrete closed surface to define the boundary between the interior and the exterior of the object. Those surfaces are decomposed of faces, as mention before. Faces can have an arbitrary number of edges each, however, surfaces are usually built out of triangles and/or quadrangles. These models are also called *polygon meshes*. Figures 1.9a and 1.10b show examples of surface triangular meshes;

Volume meshes

Using a set of connected simple volumes, such as tetrahedrons and/or hexahedrons. These models are also called *polyhedron meshes*. Figure 1.10c shows a cut in a tetrahedral volumetric mesh of a fan blade foot.

Discrete geometric representations are simple. However, they are not suitable for the up-stream design phases of a PDP for the following reasons:

- They are approximate representations that imprecisely capture the designed concept, as they fail to accurately define smooth curves and surfaces that are mandatory for manufacturing processes. Powerful shape modeling algorithms are not available in CAD systems;

its faces.

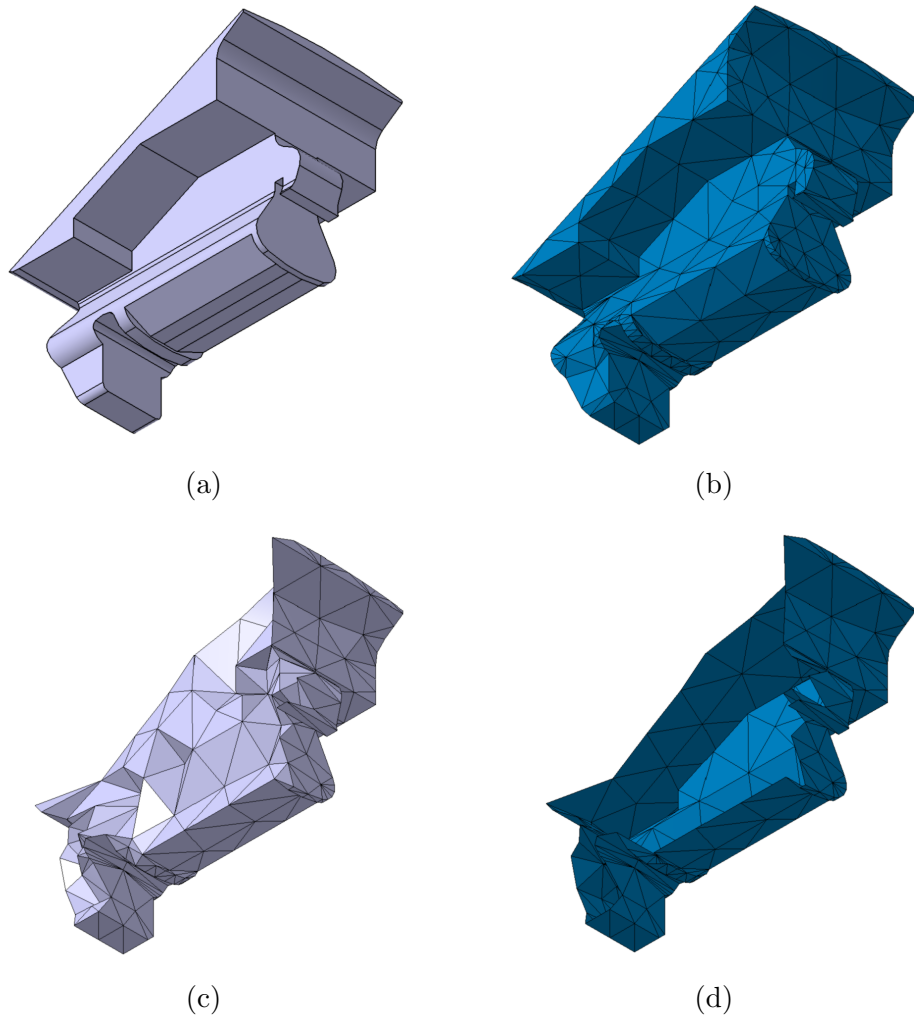


Figure 1.10: Geometric models of a mechanical part (foot of a fan blade): (a) complete B-Rep analytical model; (b) complete discrete model; (c) a cut into a volume mesh showing tetrahedrons internal to (b); (d) a section into a surface mesh showing that the triangles lie on the surface of the solid.

- Meshes are scale dependent, their level of details, i.e., their roughness, cannot be adjusted to obtain smoother shapes once the model is generated;
- The roughness of these models hinder their utilization for machining purposes in the down-stream process, where smooth realizable surfaces are expected unless the roughness is lower than that of the manufactured surface. This constraint however requires a too large amount of storage to be used for complex products.

As a result, geometric modelers of CAD systems are rarely discrete, although discrete models can be generated from analytical ones for applications in finite element simulations (see Section 1.9) and prototyping.

1.5.3 Analytical geometric models

Analytical geometric models use the same concepts defined for discrete models to describe the topology of their B-Rep model, i.e., vertices, edges, and faces. However, this topological representation is associated with geometric models such that edges need not be linear, and faces need not be planar anymore in these models. This allows for a concise yet precise representation of smooth piecewise curves and piecewise surfaces.

While vertices still represent points in the euclidean space, an edge is only partially defined by its two endpoints, since it is also characterized by the curve on which it lies. To this end, curves are represented mathematically, either as canonical geometric shapes such as lines and conic sections, or as parametric equations such as B-splines and Bézier curves.

The same principle applies to faces which are characterized by the surface they lie upon, beyond their boundary edges. Carrier surfaces are also represented mathematically, either as canonical surfaces such as planes, spheres, cylinders, cones, or tori, or as parametric equations such as B-spline or Bézier surfaces, or as implicit surfaces.

Just as in discrete models, two vertices connect to each other forming an edge, a set of edges forms a composite curve, either manifold or not⁴. A closed manifold composite curve defines the boundary of a face, faces are aggregated to form composite surfaces, again they may or may not be manifold⁵. A closed, orientable surface, without self-intersection, defines a solid C while forming its boundary ∂C .

Analytical geometric models are faithful to the original geometry that a designer had in mind since they are accurate representations of a real object. They are scalable with no information loss. Those properties make

⁴Manifold composite curves connect at most two edges at each of their vertices.

⁵Manifold surfaces connect at most two faces at each of their edges, while edges are free to be decomposed into smaller ones

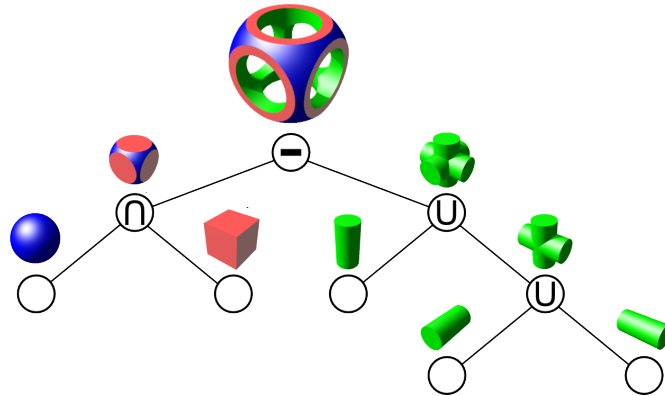


Figure 1.11: An example of CSG tree where leaves are geometric primitives, \cup is a Boolean union, \cap is a Boolean intersection, and $-$ is a Boolean difference.

it easier for geometric models to provide a reference for processes located down-stream with respect to the design process.

CAD systems use analytical representations of objects because of these favorable properties [120, 121]. Geometric modelers represent analytical models in one of following ways:

Generative methods

Where the object is defined by the process of its generation. One such method is the Constructive Solid Geometry (CSG) whereby an object is represented as a tree whose root is the geometric object and its leaves are elementary geometric entities, i.e., primitives or geometric objects generated by other generative methods, and internal nodes are geometric Boolean operations, i.e., union, intersection, differences. Other generative methods are sweeping, rotation, extrusion, etc. that generate 3D entities out of 2D sketches.

These models are useful to describe products because they keep track of a history of their modeling process and because they allow easy modifications. Geometry modifications are frequent during a product development process, hence the long lasting interest of CAD systems in history trees. Figure 1.11 shows an example of a CSG tree and corresponding geometric shape at each step of its construction;

Descriptive methods

Such as B-Rep, where the object is defined by its boundary. This object is represented by a set of closed, oriented, non-intersecting surfaces that forms a multiply connected volume. These models keep no track of the construction process, however their data size is smaller

than that of their CSG counterparts. Figure 1.9 shows an example of a teapot analytically represented by its boundary.

However, B-Rep models are automatically generated from any CSG description: each CSG boolean operation has a corresponding B-Rep transformation that represent the results in the B-Rep description. Actually, hybrid model are used in the DMU context where different B-Rep representations are linked: the CAD B-Rep model is the “exact” representation of the geometry, the visualization model is a 3D mesh approximating surfaces of the CAD model for user interaction, while FE mesh model is used for physical simulation using the finite element method.

Parametric and feature based methods

Which add geometric parameters and shape feature semantics to the primitives of CSG representation and its resulting B-Rep model. Feature-based and parametric model represents the history of the geometry construction process with a tree where leaves are parameterized shape features having shape characteristics and often fonctionnal and/or manufacturing significance: holes, chamfers, pads, pockets, blends, fastners, etc. Features associate properties and parameters to a set of topological and geometrical entities of the B-Rep model and a CSG primitive shape:

- geometric properties (dimensions such as hole diameter, 2D sketches, extrusion direction, revolution axis, sweeping curve, blending radii, etc.);
- application-specific properties (machining tool parameters, tool-path, weld beads, threading parameters, glued surfaces).

Many feature descriptions have a significance for several applications, e.g. a revolution cut can have a functional meaning (location of a bolted joint), a manufacturing meaning (drilling process), an assembly process meaning (fastening process), or a simulation meaning (definition of boundary conditions for FEA simulation). Unfortunately, manufacturing and fonctionnal properties are often missing in feature-based models due to various reasons: the time required to describe functions with features is often too long, manufacturing and functional features available in STEP ISO-10303 standard [7] and implemented in commercial software cover a small part of configurations.

1.6 DMU as an assembly

Products functionalities are satisfied by mechanical components that are assembled together to function consistently with respect to each others.

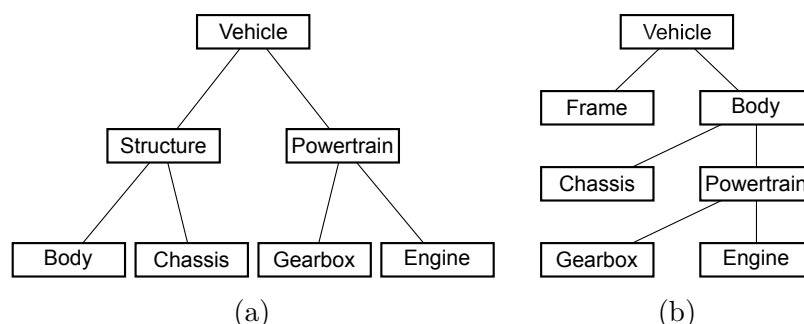


Figure 1.12: Two possible simple structures of a car DMU: (a) Assembly tree organized as per function; (b) Assembly tree organized as per order of mounting.

The DMU reflects this grouping by representing the product as an assembly of parts, each representing one mechanical components. This grouping can occur at different levels to form a tree structure, as components are gathered in sub-assemblies.

1.6.1 DMU structure

This multi-level organization gives the assembly a tree-like structure for which the root is the product, nodes are sub-assemblies, and leaves are components. We note that if generative methods are used to model components, the latter are also represented in a tree-like structure in CAD systems, with leaves being the geometric primitives and nodes geometric construction operations (see Section 1.5.3).

The hierarchical organization of a DMU using an assembly tree structure is not intrinsic to a product, rather it depends on the criterion used to set up the tree structure, e.g., functional, organizational, or assemblability. This criterion is user-defined and the tree structure is defined interactively by the user.

Functional criteria Components may be grouped according to their functional contribution to the product. In this case, sub-assemblies represent functional modules. For example, a car assembly may consist in a *structure* and a *powertrain*. While powertrain can be decomposed into *engine*, *gearbox*, *driveshaft*, *differential*, and *suspension* (see Figure 1.12a). Each of these denominations represent a functional grouping, a unit that satisfies a specific function of a car⁶. Functional modules in their turn consist of components interacting to fulfill a function.

⁶The decomposition is simplified from what a real car assembly is.

Figure 1.13 shows a snapshot of a commercial CAD software (CATIA V5) showing the tree structure of the DMU of a centrifugal pump shown on Figure 1.5. Sub-assemblies are organized according to their functional properties. The tree expands the casing sub-assembly (`Carter_pompe_centrifuge`) having as function to contain the fluid, inside which it expands the volute housing part (`Volute_pompe`) having as function to drive the centrifugal movement of the fluid.

Organizational criteria Sub-assemblies arrangement may also reflect criteria based on manufacturing and organizational choices rather than internal functional coherence. For instance, if different components of the product are designed and/or manufactured in different companies, those parts are likely to be separated in a sub-assembly, even though they do not constitute a valid functional unit on their own. Figure 1.14 shows how the aircraft structure of an Airbus A380 is divided into sub-assemblies, each being manufactured at a different facility, possibly in a different country.

Assemblability criteria Another aspect that can be encoded in a digital assembly structure is the mounting sequence of components alongside the assembly line. In this case, a sub-assembly represents a set of elements, i.e., components or other sub-assemblies, that are put up together at once. The depth of hierarchy represents the order in which installation occurs. For instance, while *chassis* and *powertrain* are two different sub-assemblies of a car, powertrain itself is decomposed into *engine* and *gearbox* whose components are mounted separately, and at an earlier stage than the assembly of the powerengine (see Figure 1.12b).

It is worth mentioning that no matter what criterion is used to organize a DMU structure, this knowledge is still partial and unreliable. This is not only the subsequence of lack of norms and standards, but also because the strict tree-like structure is incapable of representing certain semantic groupings such as functional clusters where overlapping sets may occur, or kinematic chains where cyclic graphs are expected rather than a tree structure.

1.6.2 Components' positioning

In real configurations, components are positioned relatively to each others through contacts and other assembly techniques, e.g., clamping and welding. In a DMU, however, the product is represented as a geometric model, and its components as digital solids (see Section 1.5). Contacts lose their physical meaning, welding and gluing are rarely represented, and some other unreal-

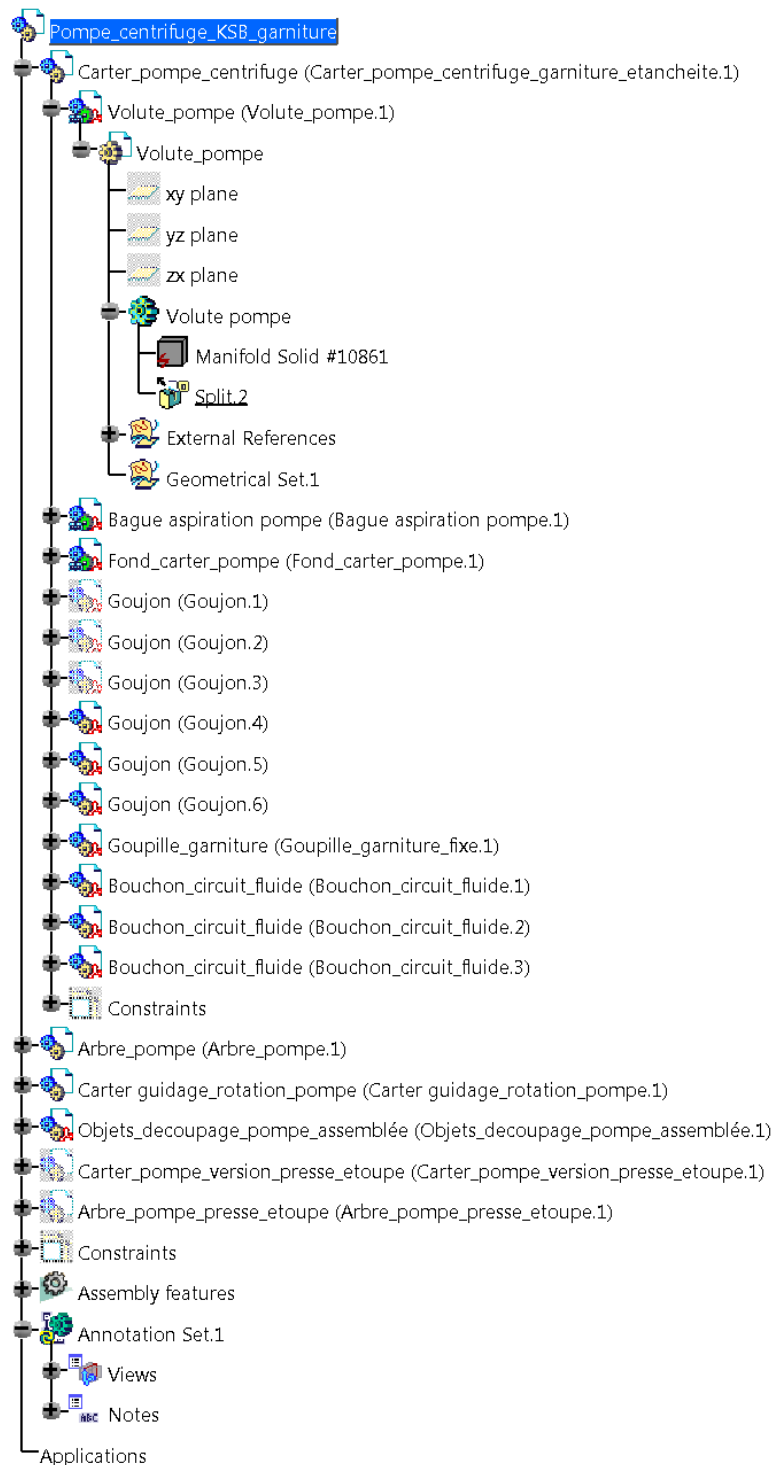


Figure 1.13: A snapshot from a commercial CAD software (CATIA V5) showing a DMU as its geometric representation (see Figure 1.5), alongside its tree-structure.

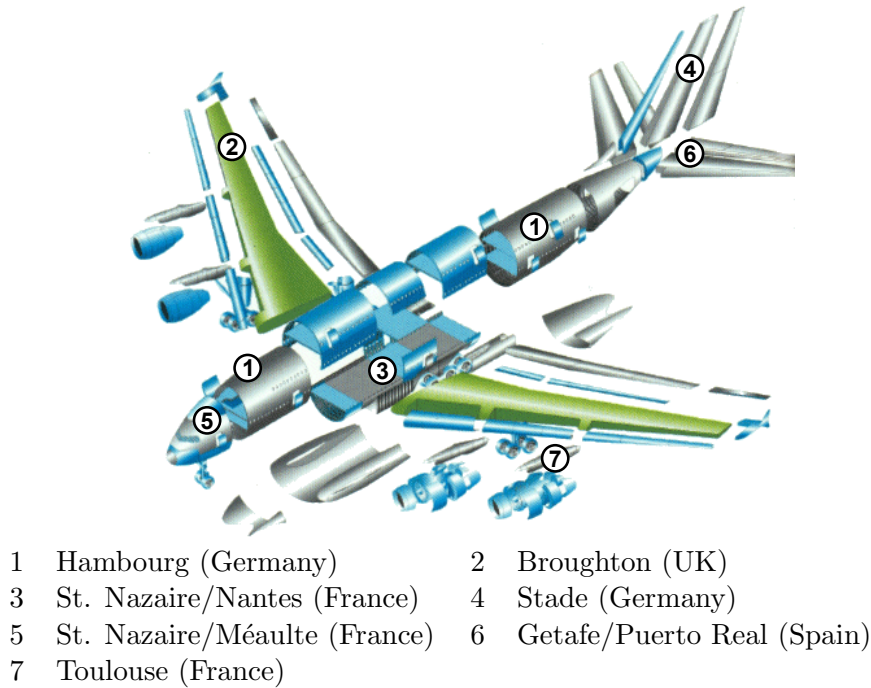


Figure 1.14: Airbus A380 airframe sub-assemblies, not organized according to their functions (cockpit, cabin, wings, tail, etc.) but according to the place where they are built (courtesy EADS).

istic geometric configurations, such as interferences, may also take part in a DMU (see Section 1.5.1).

Therefore, relative positioning should be represented in different means to convey this attachment. A set of welded or glued parts forming a single component, for instance, are usually represented as separate components since they are manufactured separately prior to the welding or gluing process. In this case, a simple contact that would represent the welding or gluing zone in a DMU is not enough to assert that components are fixed in position with respect to each others.

This first observation shows that pure geometric information to assess connection between components is ambiguous. Further, this ambiguity is often not removed in a DMU because the complementary information needed does not exist in the DMU. It is up to engineer to interpret the DMU and derive to correct contact information.

Now, concentrating on the purely geometric information related to the spatial position of components; let us carry on the analysis of a DMU content. Figure 1.16 shows a model of an internal combustion engine, with a section cut in the combustion chamber showing how the piston (the green object) fits in the cylinder of the chamber. It also shows how the piston

links to the crankshaft (the blue object) through multiple pivot links.

Different ways can be used to represent components' positioning and orientation, depending on what to expect from the DMU.

Kinematic links

For kinematic simulation purposes, where animations of the mechanism are to be generated, kinematic connections define relative positions between objects, leaving some degrees of freedom (DoF). Simple connections, also referred to as *lower pair connections* are classified into *prismatic*, *cylindric*, *screw*, *planar*, and *spheric* connections [83]. In the example of the internal combustion chamber, a sliding pivot joint between the piston and the cylinder, which is a cylindric connection, aligns both axes leaving two DoF: rotation around and translation along the common axis⁷. The connection between the connecting rod and the crankshaft is defined by a pivot link that allows only a rotation around their common axis B . Figure 1.15 shows the kinematic diagram of the mechanism of a slider-crank.

Geometric constraints

Manufacturing models give more importance to how components are located with respect to each other, rather than what relative motions they exhibit. Thus, relative positions of components can be defined through geometric constraints such as coincidence, concentricity, coaxiality, distance, etc. These constraints usually leave the object stationary, i.e., with zero DoF. Constraints that are not defined by the designer are assumed by the modeler, usually in terms of linear and angular distances, leading to a static representation of the product.

In the example of the combustion chamber shown in Figure 1.16 a coaxiality constraint is defined between the piston and the internal cylinder of the chamber. The distance between the piston and the back end of the chamber is either defined or assumed.

Absolute positioning

Another way to create the scene of an assembly is to directly position and align objects according to a global coordinate system defined assembly-wise. This is usually achieved using an affine transformation matrix per object.

The latest approach is the simplest, though it has many disadvantages:

- Setting the parameters of transformation matrices during design is cumbersome. Designers usually define their concepts by means of kinematic or geometric constraints;

⁷Rotation is eliminated when considering other connections in the mechanism.

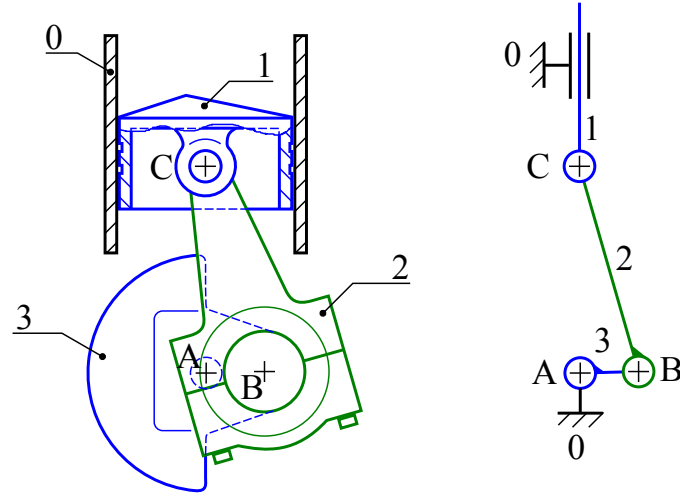


Figure 1.15: A drawing of a slider-crank mechanism (left) and its corresponding kinematic diagram (right): (0) Chamber; (1) Piston; (2) Connecting rod; (3) Crankshaft. A, B and C are points locating the rotation axes of the crankshaft, connecting rod, and piston, respectively.

- Editing of the assembly model is more complicated, since constraints propagation should be considered manually now and the consistency of the assembly model is harder to achieve;
- Kinematic links are indicators of relative motion properties between components. This information is lost when fixing components in place in an absolute manner.

Nevertheless, standardized formats describing components and assemblies across CAD software, such as STEP [7], still use absolute positioning of assembly components, as it is globally recognized by all geometric modelers. Geometric modelers, in their turn, allow the exportation of models in such formats, even though native models usually use one of the first two methods, or a combination of them to position and align components.

While kinematic links explicitly characterize the relative motion between components using rotations axes and sliding directions, geometric constraints are meant to fix components in place, generating an instantaneous representation of the product at a given moment in time. Another distinctive difference between these two methods is that geometric constraints, as per definition, must refer to explicit geometric entities of components, unlike kinematic links, such as helical motion, where certain attributes, such as pitch, can be provided as a parameter, independently from any geometric support. Other kinematic links, e.g., pivot link, do not need to refer to geometric entities of the components, e.g., the cylindrical surfaces of the pivot

link, which means that the existence of a pivot link does not mean that the corresponding components are geometrically consistent, i.e. of same diameter. Considering that a product is subjected to numerous modifications during its design process, this shows that kinematic links alone are not a reliable source of data to ensure that a DMU is consistent with respect to the relative position of its components.

Moreover, geometric constraints are of minor significance to design intentions, since designer may use misleading geometric alignments for the sake of ease of use. An example would be the alignment of the piston with the chamber in Figure 1.16. One may assume that the contact between the surface S_p of the piston and the internal one S_c of the chamber could be inferred thanks to a coaxiality constraint, plus an additional diameter check. However, this conclusion is not always achievable as the user-defined positioning can be obtained by aligning, say, the axis A_p of S_p to the axis of any given external cylindric or conic surface, S_{cy} or S_{co} respectively, of the chamber, leading to the same exact geometric configuration. Kinematic links in turn bear an inherent sense in the context of motion simulation and analysis. They are, however, often disconnected from boundary geometric elements. For instance, if kinematic links are used in the example of the combustion engine, the piston can be linked to the chamber by means of a sliding pivot link. To establish this link, cylinder axes A_p and A_c should be used. Such a link makes no reference to concrete geometric entities such as boundary surfaces S_p and S_c (thus no reference to contact zones) and leads to no avail when geometric interactions detection is sought.

As a conclusion, this renders reasoning based on geometric constraints or kinematic links an unreliable approach. Globally, none of the three methods to position components in an assembly is, alone, sufficient to ensure the geometric consistency of an assembly model.

1.7 Other information associated to a DMU

So far a DMU is regarded as a set of geometric objects (components) grouped together in a hierarchical structure (sub-assemblies). This representation incorporates geometric information about the product, plus some kinematic properties as seen in Section 1.6.2. However, and in order to efficiently participate to the PDP, the DMU has to integrate other information about the product and its components rather than its geometry, kinematic links, and assembly structure. This information adds up to the geometry of components, but is not part of it.

One reason that this “extra” information is needed is the fact that the geometry of the product is often simplified in a DMU (see Figure 1.17), thus differing from the shape of physical components. Standards as well as companies’ practices suggest to compensate this loss of information due to

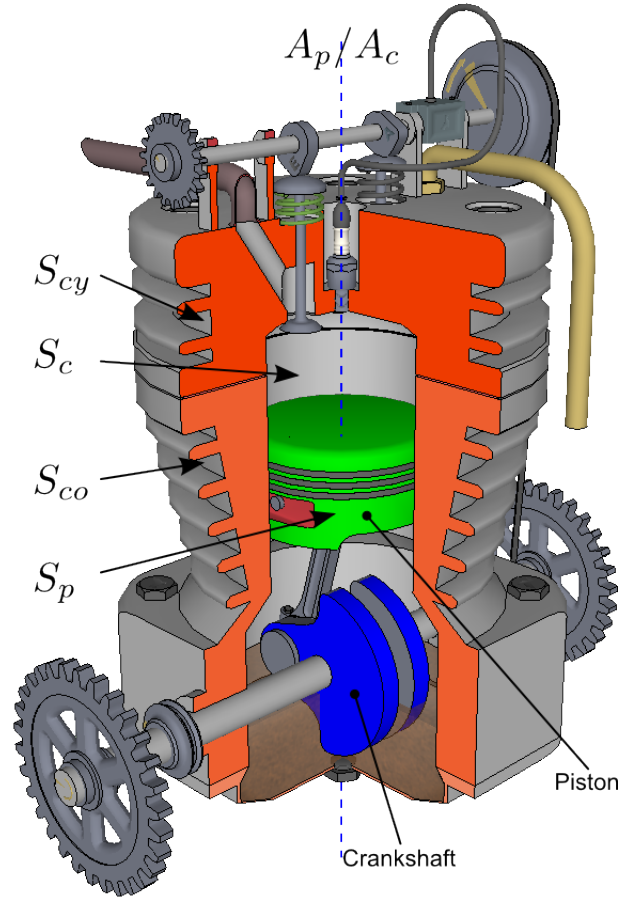


Figure 1.16: A DMU of an internal combustion engine. A section cut in its combustion chamber shows the piston (in green), and the crankshaft (in blue).

geometric simplification by other means of auxiliary annotations. Figure 1.7 shows an example of information lost due to such a simplification. The ISO standard SETP AP214 [12], for instance, provides annotations such as *thread*, where geometric properties such as *major diameter*, *minor diameter*, *pitch*, *number of threads* and *hand* can be expressed explicitly [69]. This allows the threaded part of a screw or a nut to be geometrically simplified, e.g., as cylindric surface.

Another kind of such necessary annotations in the context of a PDP is important information that cannot be represented geometrically. An example is component material and its physical properties. Though it relates in more than a way to geometry, kinematic knowledge about the product also falls into this category, since relative motion holds more information than



Figure 1.17: Screw and nut (a) in real configuration, (b) as simplified geometric model.

instantaneous shape snapshots.

Integrating materials and their properties in the DMU is necessary to enable the generation of detailed bills of materials (BOM), which design office communicates to other departments such as purchase department [74]. BOM contains detailed information about required parts and material to enable the manufacturing of a product [165]. This information is used to prepare orders and manage inventory.

A close look at industrial practices shows that this information is scattered around the DMU in a non-standardized manner, making the task of exploiting such knowledge a challenging one. Iyer et al. [90] show that modern Product Lifecycle Management (PLM) systems provide the DMU with a context that allows annotations such as keywords and part name, etc. Authors, however, show that these annotations are not robust, and of little use for information retrieval. They attribute this inadequacy to reasons among which are the non-unified conventions among design personnel, and the change of industrial context with time.

1.8 Application of DMU

DMUs are computerized models that engineers use to communicate their designs to the manufacturing as well as other company departments. Their obvious application then is to provide the pattern upon which the product is to be built.

However, they contain enough information that allows engineers to use them at other stages all along a PDP.

In the previous section we saw that a DMUs contains supplementary annotations that allow the generation of reports, such as BOM, necessary for inventory and purchase management.

Another important application of DMUs is the generation of simulation models. Since they closely represent product geometry as well as other phys-

ical properties, DMUs are good candidate to extract simulation models out of them. Extracted models differ according to the goal of the simulation, we can recognize structural simulation models, thermal simulation models, fluid simulation models, among many others. A prominent simulation method in todays PDP is the finite element method, Section 1.9 sheds more light on this method. Auxiliary annotations, particularly kinematic connections between components, contribute to a special type of simulations that are based on the content of a DMU, this is Virtual Reality (VR) simulations.

Some VR simulations have their applications in high risk environments when testing and training is too dangerous to be performed in real environments. In such cases, a DMU can be used to generate a simulation model where the desired tasks can be conducted in a completely virtual setup.

As demonstrated in Section 1.6 DMUs also provide a structure that group components into subassemblies that are then grouped themselves into an assembly representing a functional product. In this context, DMUs play also an important role in the assembly/disassembly planning process of PDP.

Considering its diverse applications, a DMU shows a prominent presence at different stages of todays PDP.

1.9 Basic principles of finite element analyses

The finite element method has shown its merits in different simulation applications. In this section we introduce, in a nutshell, the basic concepts of finite element analysis (FEA).

1.9.1 Numerical approximations of physical phenomena

FEA is a numerical method in which certain physical behavioral phenomena are studied and analyzed using numerical approximations of real objects called FEMs. A FEM contains a discrete representation of objects' materials in space, achieved using one-, two-, or tree-dimensional meshes (see Section 1.5.2). Information about material physical properties are assigned to the mesh at the element level. An element can be an edge, a polygon or a polyhedron.

Figure 1.18 shows an example of the results of a FEA on a pump casing to study heat conduction.

FEA simulations are in the heart of modern PDPs and product validation practices. The FEM is prominent in most of behavioral studies of a product prototype.

The process of FEA consists in three general steps:

pre-processing In this phase, the FEM is generated. This can be done from scratch, building the mesh model at first, then assigning it physical attributes. Nevertheless, in today's industries the CAD model,

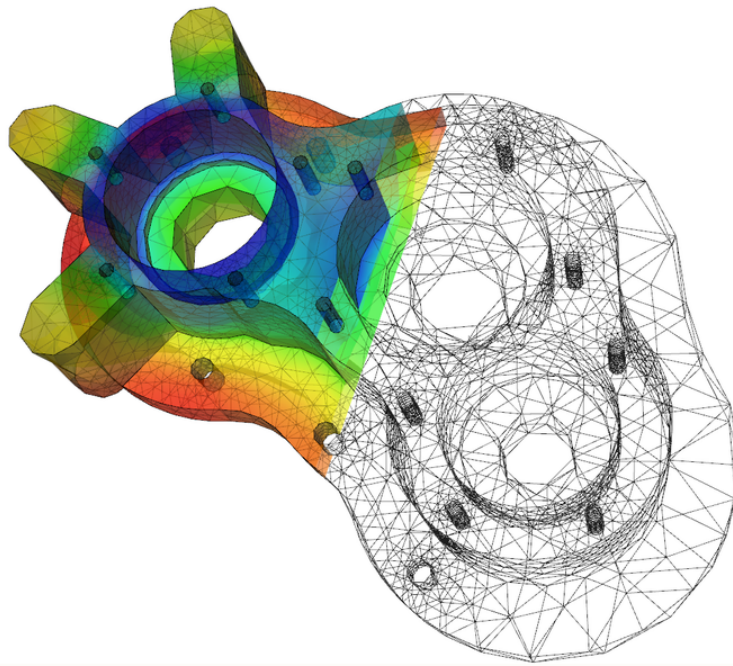


Figure 1.18: A FEM of a pump casing showing the results of a heat conduction analysis.

which is usually an analytical one (see Section 1.5.3), tends to be integrated with a mesh to facilitate the propagation of modifications between these models. Then, the mesh acquired is in turn enriched with material properties, in order to obtain the FEM. This process is not as straightforward as it may seem, since many factors affect the quality of the generated model, thus results' accuracy when an FE model can be effectively obtained. Such factors are how close the approximate model, i.e., the meshed shape, is to the original objects; how many elements does the model have; what are the distribution and the quality of those elements over the original domains, etc.

analysis Now that the FEM is produced, the simulation problem is solved by dividing it into simpler ones at the FE level, using differential equations with boundary conditions. The type of equation used depends on the desired simulated phenomenon. For instance, while Euler-Bernoulli beam equations are used for structural simulations [82], heat diffusion formulas are applied for thermal behavioral studies [45], and Navier-Stokes equations for fluid simulations [18]. The solution boils down to an error function minimization problem, respecting boundary conditions.

post-processing The analysis phase comes out with observed variables values over the meshed domain, i.e. solution fields. Those fields represent studied changes in physical properties such as displacements, temperatures, etc. To provide a global overview of the underlying simulation, those fields can be visualized using color codes, that allow engineers to better estimate zones of interest.

In this work we are only concerned about the first step; the pre-processing.

1.9.2 Generation of a FEM

Pre-processing has a crucial impact on the efficiency, performance, and accuracy of later steps of FEA. Many choices are made at this stage such as shape simplifications, mesh element dimension (linear, surface, or volume) and mesh element shape, e.g., triangular, quadrangular, tetrahedral, hexahedral, etc.

The resulting FEM highly depends on the observed phenomenon and the ultimate goals of the FEA. To this end, simulation *objectives* should be outlined first, such objectives can be either structural study, with deformable bodies, or thermal behavioral analysis, etc. Once objectives are defined, assumptions about the relevance of geometric areas can be made, leading to simplification *hypotheses* characterizing some areas as details with respect to the simulation objectives. This enables the reduction of models' complexity through shape transformations [41]. Entire shapes of objects or a small subset of them can either be simplified (case of dimension-preserving transformations, e.g., hole removal from a volume that transforms a volume into a new one), or idealized (case of dimension-reducing transformations, e.g., replacing a thin and elongated volume with a beam that transforms a volume into a line).

The shape transformation process generates what we refer to as *simulation model*, also called mechanical model in the literature [168, 21, 80]. Alike the DMU, the simulation model contains a geometric representation of the assembly that is dedicated to simulation purposes rather than manufacturing ones. Such differences refer to the concept of product view where the simulation view is distinguished from the design one (see Figure 1.19). Along with the simplification hypotheses and simulation objectives, the simulation model provides essential knowledge to generate the required FEM. Figure 1.19 depicts how, for a given simulation context, geometric transformations generates a simulation model out of an assembly DMU, whose mesh is then generated to produce the FEM [41].

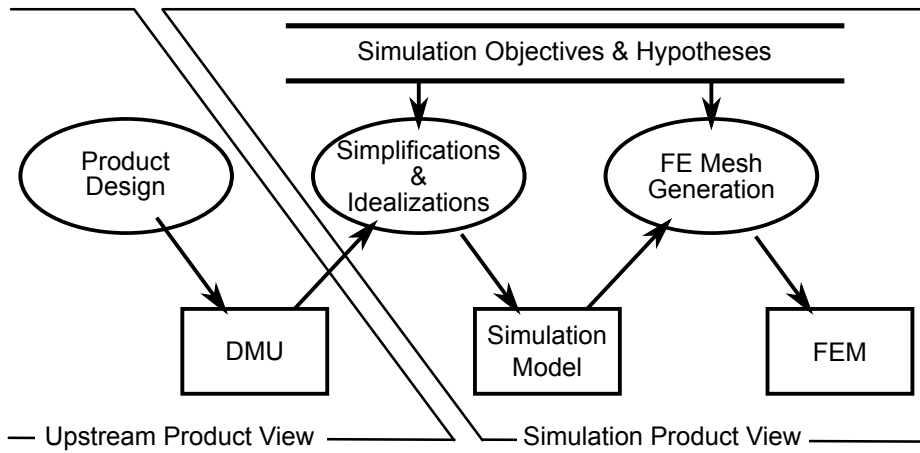


Figure 1.19: A flowchart showing processes and models involved in the preparation of a DMU for FEA. It locates the simulation model as a result of the simplification and idealization processes, and as an input of the FE mesh generation process.

1.10 DMU as polymorphic representation of a product

Different applications of a DMU require different types of information, and different levels of details, particularly when geometric representations are included in these applications [71].

For instance, while purely volumetric models are recommended for manufacturing and machining applications since they most accurately represent the real shape of components, reduced-dimension configurations are tolerated, even recommended, for simulation-oriented applications where geometric details become a burden and shape simplicity is prioritized over its accuracy to promote the accuracy of the simulations results.

In the same context, GD&T information is vital for manufacturing department, while relative motion properties are usually irrelevant at this stage. However, such kinematic knowledge is essential for rigid body simulations where geometric details such as manufacturing tolerances are meaningless.

This diversity of applications requires the DMU to show different *forms*, or views, according to the level of details that the application implies. In this sense, the DMU acts as a polymorphic representation of the product, where the shapes of components as well as their associated annotations are dependent on the nature of the application.

1.11 Adapted definition of DMU

The concept of DMU has been developed across the literature, and many works have tried to establish a definition. Those definitions all agree that a DMU is a digital representation of a product that contains at least its 3D geometry down to a certain level of details. They also assent that it plays a major role in the PDP. However, they differ in defining to which extent a DMU participates to certain stages such as simulation and validation.

Some definitions extend the concept of DMU to the point where it incorporates all virtual prototyping, referring to simulation models as part of the DMU since they are indeed a digital representation of the product [57, 177].

Other works restrict DMUs to models that fit directly the purpose of design and manufacturing. While simulation models can be generated based on DMUs, these simulation models are not effectively part of them.

In our work we adopt the following definition.

Definition 1.3 (Digital mock-up). A DMU is a computerized representation of a product as an assembly of sub-assemblies and components in a hierarchical structure. It represents product geometry, possibly at different levels of details. DMUs also contain supplementary information about the product and its components that is casted in compliance with the intended application.

This definition puts forward the polymorphic nature of a DMU since geometry and other associated information are adaptable to the application in which a model fits.

It is worth mentioning that although the concept of DMU covers a wide variety of functional, kinematic and technological informations [69], only few efforts are made to standardize non-geometric data. Therefore, in the scope of the present work, we are only considering the geometric information that a DMU holds, where components are positioned absolutely according to a global (product-wide) coordinate system as it is the case for standardized formats such as STEP. Other information such as kinematic links, technological annotations, etc. are not considered because they are ambiguous and unreliable.

1.12 Conclusions

DMUs accompany PDP and are now much more than manufacturing issues as analyzed in Sections 1.8 and 1.9, and particularly in Section 1.9.2. Indeed, DMUs have become focal to modern PDPs as they represent a central data repository of a product incorporating models as generated by the engineering design office.

Further than a product model, DMUs contribute nowadays to the generation of virtual prototypes that feed numerical simulations to assess product functionalities (see Section 1.9).

This data repository is organized around a geometric model that forms the kernel of a DMU (see Sections 1.5 and 1.6.1). However, considering the geometric interactions between components of a product, as they are represented in a DMU, Sections 1.5.1 and 1.6.2 have shown that the digital shape of components may significantly differ from that of physical ones and component positioning techniques do not ensure the consistency of an assembly. Engineers' know-how refers to various conventions and standards to interpret component shapes as well as their relative positions. These conventions and standards are not part of the CAD systems used to produce and modify the geometry of DMUs. In addition, the diversity of processes, e.g. FEA, assembly simulations, requiring a pre-processed DMU as input leads to the concept of polymorphism (see Section 1.10).

Complementary information can be attached to components and/or sub-assemblies (see Section 1.7) or derived from assembly models (see Section 1.6.2) but it appears that the robustness and reliability of these informations do not strengthen a DMU and, in addition, are not easily accessible. Among all these informations, it is important to mention that functional information is not explicit in a DMU and consequently, there is no effective link between any description of a function and some corresponding geometric entities in a DMU.

In the present work, the focus is placed on the generation of explicit functional information attached to components and sub-assemblies. The attachment of such information to geometric entities is of strong interest since the polymorphism of DMUs is a key requirement to pre-process them and generate virtual prototypes. Among the virtual prototypes that can take part to a PDP, the structural behavior of a product is of increasing importance. Therefore, the shape transformations taking place during the pre-processing for FEA is of strong interest, especially in the case of assembly behavior. Relating shapes, i.e., the geometry of assemblies, to component functions in order to ease the assembly pre-processing for behavioral simulations is a consistent objective addressed here. To this end, further analyses of prior work that relate shape, function and behavior of a product or, otherwise, address either of these concepts separately is the purpose of the next chapter.

Chapter 2

Literature Overview

This chapter analyzes existing literature that relates to this work. Work that founded the conceptual or methodological basis of our approach is presented in Section 2.2 that studies product function and Section 2.3 that relates function to shape. Later sections study prior work that addressed problems of interest to our research. Section 2.4 visits works that aimed at the extraction of functional information out of geometric data, showing the important role that components interactions play whenever functionality is sought at the assembly scale. Section 2.5 then studies efforts to define components interactions geometrically. The problem of knowledge representation and its importance to the DMU is addressed in Section 2.6, and works that tackled this problem from different angles are reviewed. Finally, Section 2.7 examines works dedicated to the transformation of CAD models to FEA-friendly simulation models, emphasizing the importance of functional knowledge at this stage. In the aforementioned sections, we also show why already established work failed to provide satisfactory answers to some of the difficulties observed in the previous chapter and hence, why they motivate our work.

2.1 Introduction

As mentioned in the introduction of this document, the upcoming chapters will concentrate on formulating the problematic that motivates this work (see Chapter 3), before relevant concepts are defined (see Chapter 4) and the proposed method is detailed (see Chapters 5, 6 and 7). However, and before proceeding likewise, a review of what state of the art offers is imperative. The objectives of such a literature review are twofold:

- The major objective of the proposed approach focuses on enriching assembly models with functionally related information, as Chapter 1 has

shown that the content of a DMU ends up as a simple geometric model without functional information. It is therefore mandatory to review concepts such as product and/or component functionality and how these concepts can be related to geometry, i.e., product and/or component shape. To this end, design process studies and corresponding design methods that advocate the function–behavior–form relationships, a central paradigm to this research, need to be analyzed to evaluate how a geometric model of an assembly can be related to its functional information. One of the outcome of this analysis shows the prominence of geometric interfaces between components;

- Works that associate component geometry with functionally related information such as feature-based approaches, detection of geometric interfaces between components and knowledge-based approaches are also studied since they are part of prior approaches that proposed concepts related to function. Because a particular focus is placed on the use of functional information in the context of FEA, some key approaches that relate to FEA of assemblies and geometric interactions between components are also reviewed to better highlight the challenges in this area. Not only the common denominators with such works are put forward, but also differences that distinguished the approach proposed in this document and made it worthwhile. To the author’s knowledge, none of the existing works addressed and efficiently solved the tackled problem of functional enrichment of DMUs for FEA purposes.

Such a literature survey prepares the ground for the discussion to come in the successive chapters, and situates them with respect to existing works. Section 2.2 starts this review by showing how functionality is seen from different angles in the domain of mechanical engineering.

2.2 Function formalization

Although the concept of *function* may initially seem self-explanatory, literature has different points of view regarding its definition. Deng et al. provide an overview of different perspectives [62].

Three distinguished standpoints can be identified, from which a function is seen:

as a *raison d’être*. A function is defined by many scholars from a teleological point of view as the ultimate purpose of a product [60, 29, 46, 84, 136, 163, 172, 178].

An interesting definition to our approach is that of Gero [72] as he defines function to be *the semantics of design*.

as a black box. Others considered function as the relationship between the input and the output of a system [105, 129].

as a verbal phrase. Function has also been regarded from a performance perspective as it defines the behavior of a product [174, 172, 77, 85, 167, 162].

Literature suggests modeling functions as verb-object pairs [129, 86], such as ‘reduce speed’ or ‘transfer torque’.

Many scholars, however, saw functionality from a midpoint perspective between two or more of the above-mentioned ones. Pahl & Beitz [129] formalized functionality as a verb-object pair, in which the verb expresses the function, while the object represents the flow of material, energy, or signal between the input and the output of a system. Likewise, Qian & Gero [136] describe a function as the purpose of design, while emphasizing its strong ties with behavior.

The above-mentioned works viewed functionality independently from any product state. Without going into the details of the meaning of a product state, it can be observed that a state can be related to the input and output of a product. Indeed, a state can be characterized by a set I of physical input parameters and another set O of physical output parameters by the product. I and O are subsets of the whole set of input and output parameters, respectively, that are used to describe all the possible configurations the product can reach during its conventional usage. As pointed out above, input and output parameters of a product are means to define functions through the couples of input, output parameters that a function binds. It is clearly the case when authors consider a function as a ‘black box’, which is a casual standpoint, but also when authors concentrate on a behavior since its purpose is to take parameters as input and modify them to produce output parameters, which clearly underlines the concept of state. When modeling functions through verb-object pairs, the verb expresses also an action, which refers also to the concept of behavior. Therefore, a state can be seen as a collection of functions that pertain to the sets I and O of physical parameters attached to a product.

Therefore, the two concepts are closely related to each other, as the products deliver different functions at different states. For instance an *operational* bike delivers the functionality of transportation, while a *dismantled* one has a better mobility which makes it easier for shipment. A broken bike, however, delivers no particular function at all.

If states and functions are related to each other, it can be observed also that the proposed definitions of a function do not refer explicitly to the shapes of the product or some of its components.

In this work *function* is considered to be the major motivation behind product design, with strong connection to the product state as part of its

design. Now that the concept of functionality is well-situated with respect to different literature viewpoints, the next section will develop on how to link this concept to product and component forms. To this end, as the upcoming section will reveal, it is essential to address the behavior of a functional entity.

2.3 Connections between form, behavior, and function

The link between form, behavior, and function appeared early, and has been discussed exhaustively, in the literature of engineering. De Kleer [60] intuitively defined function as what a device *is for*, behavior as what a device *does*, and structure as what a device *is*. When applied to mechanical engineering, and from a design point of view, structure, under this understanding, maps to form.

In an effort to formalize the relationship between function, behavior and structure, Umeda et al. [174] established the Function-Behavior-Structure (FBS) diagram, with a strong emphasis on a behavioral understanding of functionality. Structure from the authors' perspective denotes physical attributes of an object. This can be seen as a generalization of object form.

2.3.1 Behavior to complete the design puzzle

Design is seen by scholars as a goal-oriented activity that aims at satisfying certain requirement expressing a desired functionality. To this end, the link between function and form, particularly in the domain of mechanical engineering, is to be established. However, no direct mechanism allows for that matching. Gero [72] shows how function can be formulated into an *expected behavior*, and how form can be analyzed to produce its *structural behavior*. This reduces the design activity to the process of matching expected and structural behaviors, either through the evaluation of existing forms or the synthesis of new ones.

Qian et al. [136] outline the casual relationship between structure and behavior and between behavior and function that allows a product to meet its expectations. This relationship is shown to be bidirectional, as function can be analyzed to infer potentially-multiples behaviors that lead to its satisfaction, then potentially-multiple structures can also be inferred in what is referred to as goal achievement paths.

2.3.2 Pairs of interacting interfaces

In Albers et al. [23], the authors do not only emphasize the connection between product geometry and functional attributes, they also demonstrate

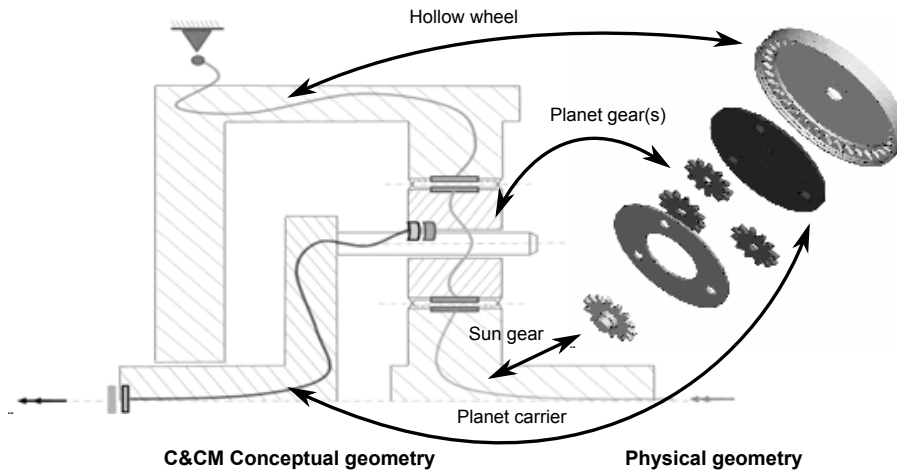


Figure 2.1: An example of a planetary gear modeled using C&CM [23], showing interfaces between components of the corresponding assembly.

with concrete examples the correlation between pairs of interfacing geometrical entities, i.e., pairs of surfaces belonging to different components, and the expected purpose of a product. The corresponding design methodology shows, through industrial case studies, how a function is tightly coupled with the geometric properties of interfaces between neighboring components that provide the desired, or even undesired, behavior. Figure 2.1 shows an assembly model using Contact and Channel Model (C&CM) introduced by Albers & Matthiesen in [24]. The example puts forward the important role interfaces between components play in a product assembly to achieve the desired function.

2.3.3 Tools and guidelines to support the design process

The aforementioned approaches have been applied to or are part of design methodologies to provide engineers with tools to facilitate the creative activity of design.

As an example, the work of [22] builds upon the C&CM to develop a modeling approach as a tool to assist the design process.

Authors in [147] present a theoretical framework that builds upon form-function mapping to provide guidance to engineers along the design process, in an effort to automate, or semi-automate, the transition from user requirements into a functional artifact. In the latter work, authors address the relationships between function, behavior and geometry from a top-down¹

¹By *top-down*, we refer to the path from functional specification, to design attributes, particularly, components shapes.

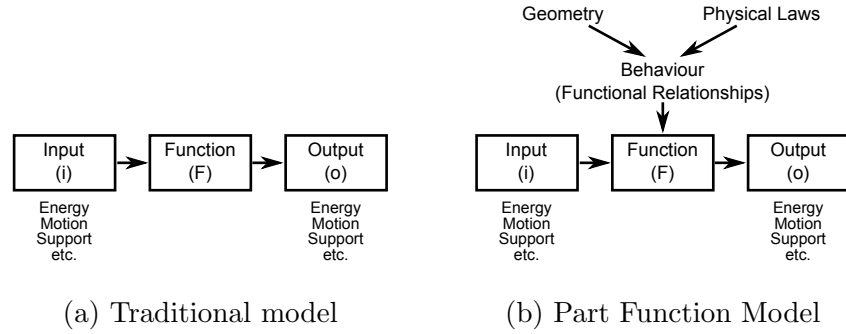


Figure 2.2: A comparison between a traditional conceptual model of part function (a), and PFM introduced by Roy & Bharadwaj [146] (b).

perspective, as a goal achievement guide to obtain parts geometry from functional specifications.

Roy & Bharadwaj [146] set up a design approach to connect functions to 3D geometry using a Part Function Model (PFM) illustrated and compared with traditional models in Figure 2.2. To acquire the proposed model, a designer should provide a behavioral description of parts using a predefined vocabulary. Along with geometrical properties of parts contacts, this description is used to infer functional interactions between components. The PFM obtained involves functional information down to the level of boundary faces since the behavior model builds upon interfaces between parts, i.e., contact surfaces. This work emphasizes the discontinuity between spacial properties of parts in an assembly model and their function, and advocates the importance of a behavioral description to connect function and shape concepts. It, however, requires the user to manually provide such a description before making any judgment about a part functionality.

At the level of complex assemblies with hundreds to thousands of parts, the amount of complementary data defining a behavioral model becomes too tedious to add. Kim et al. [103] provide a formalism that augments a DMU with the design intent of an assembly, particularly the semantics of contacts between assembly components. This is achieved through the Assembly Relation Model (ARM) and an XML-based meta-model that refers to geometric entities in the DMU. Using this formalism, assembly models are annotated by collaborating designers, based on spacial relationship analysis undertaken by a geometric kernel. Interactions between components are referred to as *joints* in the context of assembly design. The nature of interacting form features between components, then called *mating feature*, is captured in the Generic Assembly Relationship Diagram (GARD) as part of the ARM, in a graph-like manner. Joints in GARD are reduced to global parameters of welded, glued, bolted and riveted connections that define components position with respect to each other. This work is meant to facilitate collaborative

assembly design and enriches a DMU with information relative to this task.

However, simulation objectives may require a detailed representation of interfaces functional interactions beyond mere assembly joints. Especially when the purpose is to assess the stress field distribution in a bolted connection with tens or hundreds of bolts. Additionally, one can observe that the relationship between shape and function strongly relates to interfaces between components and, more precisely to contact surfaces, i.e., the real component relative position. Somehow, this relationship is not robustly expressed in a DMU, due to the designer's choices made about component shapes (see Section 1.5.1 and Section 1.7). Indeed, component shapes are often simplified in a DMU, which can alter the representation of physical contacts between components. As a result of the analysis of prior work, none of them has taken into account this discrepancy, which can be regarded as a fact creating inconsistencies between functional prescriptions of a designer and the content of a DMU.

Works examined so far share a common denominator where assemblies are described as geometric models enriched with technological annotations that may qualify as functional information reflecting design rationale [107]. They participate to the enrichment of a DMU as a central repository of PDP activities (see Section 1.7). While each of the unfolded efforts has seen the DMU from a particular standpoint, none of them satisfactorily considered the requirements of a seamless generation of simulation models. Observations showed that the closer the enrichment is to any functional significance, the higher the lack of information external to CAD environments and the greater the need of user interactive input during a design process.

As a common observation of all prior works reviewed in this section, they are all heavily dependent upon designer's input data, i.e., the consistency of functional information. Its relationship with 3D geometric models of components or assemblies are left under the designer control. This is not tractable for large assemblies and hard to set up even for simple products with dozens of components.

Although proved bidirectional [136], the function-form relationship is studied from a purpose-oriented viewpoint by the so-far analyzed works, i.e., along the path from intended function to a designed form, oftentimes through physical behavior as a connector. This understanding of the relationship remains dominant in the literature. The following section, however, shows work that made use of the bilateral nature of this relationship, exploiting the causal direction, from form to function, to develop some confidence about product functional properties.

2.4 Constructive approaches to deduce function

In this section we walk through some of the existing work that aims at the association of functional properties to different elements of a DMU. The previous section has already highlighted that shape simplifications of components restrict the applicability of approaches strictly based on geometric contacts between components, which is the common feature of prior work.

2.4.1 Form feature recognition

Efforts have been paid in the field of Feature Recognition (FR) in solid models as early as 1988 [94]. Zhu & Menq define features, also referred to as form features or machining features, to be ‘*the representations of shape aspects of a physical product that can be mapped to generic shapes in a given context and are functionally significant*’ [185]. This definition establishes links between form features and functionality, and makes it of a particular interest to our research to shed some light on FR-related studies.

Examples of manufacturing features include holes, slots, pockets, and other shapes that can be obtained by material removal operations using Computer Numeric Control (CNC) machining systems [81].

In many occasions, literature shed the light on the gap between the low-level geometric information usually present in CAD models, and the higher level functional semantics needed by CAM systems. Authors in [121] promote features as the link between pure geometry and design semantics, allowing a smooth transaction from CAD to CAM activities. Literature also surveyed a wide range of techniques that participate to the Computer Aided Process Planning (CAPP) automation as a link between CAD and CAM, where FR plays a major role as a communication agent [153, 152, 166].

Authors in [25] address the problem of functional features extraction out of digital models. They classify existing solutions into human assisted approaches, feature-based modeling, and automatic feature recognition and extraction. Han et al. [81] in their turn regroup automated FR algorithms into graph-based techniques, space decomposition approaches, and rule-based geometric reasoning.

Falcidieno & Giannini [66] suggest a three-stage solution: recognition, extraction, and organization of features. The proposed system builds a hierarchical structure of a part shape in accordance with level of details. The recognition phase builds on the work of Kyprianou [106] that paved the road of graph-based FR.

A graph representation of the geometric model of an object is generated in [94], before graph matching techniques are applied to extract form features, also represented as graphs.

Ames in [26] advocates an expert system approach to recognize application-specific features given the product solid model as an input.

In Date et al. [58], FR is integrated into the process of simplification as a preliminary step to prepare a meshed model for FE analysis.

A technique to detect and remove blending features is presented in [185]. Even though fillet and round are secondary features as they are of little functional significance (they don't actually conform to Zhu & Menq's definition) their presence may interfere with the detection process of their parent features. Authors, thus, present their work as a preliminary treatment of the geometric model to enhance the recognition of more significant functional features. Another approach, capable of handling more interacting shape features through an iterative method is presented in [175]. In this work form feature recognition techniques are used to detect features face-sets, and then a feature is removed before passing to the next iteration, where previously interfering features can be detected.

In Sunil et al. [164], authors again tackle the problem of features interaction through a hybrid approach for FR that is both graph- and rule-based.

A more exhaustive categorization of efforts paid in the area of FR is given by Shah et al. [154]. However, fruitful studied categories did not diverge much from earlier classifications [66, 25, 81]. As a complement, Shah et al. in [154], address recent work that considered the otherwise overlooked free-form features [158, 182, 159].

Sridharan & Shah [159] provide a feature classification method to aid the detection of complex CNC milling features. Figure 2.3 shows an overview of the proposed taxonomy. The preliminary recognition of features uses a rule-based approach independently of any geometric restriction, thus, allowing for the identification of features involving free-form surfaces. At this stage, features are categorized according to the first level of the taxonomy presented in Figure 2.3. A finer classification of recognized features then takes place, to predict feature type more precisely, this time taking into account geometric properties such as surfaces types (cylindric, ruled, free-form, etc.). This corresponds to the categorization of features in the second level of the taxonomy presented in Figure 2.3.

Automatic FR techniques aim at the extraction of functional information as design intentions given the pure geometric model, thus contributing to the enrichment of a DMU. They are, however, still limited to a very small set of simple geometric configurations like holes, pockets, slots, rounds and fillets. Most of prior work fits into a bottom-up² approach where features are extracted from low level geometric entities and a detached volume model is processed as a standalone entity. Whenever product models are referred, they are generally regarded as a collection of components processed with loose or no connections at all between them.

²In contrast to top-down scheme, *bottom-up* is used to refer to the path from existing design information, such as objects shapes, to a more elaborate knowledge, such as functionality.

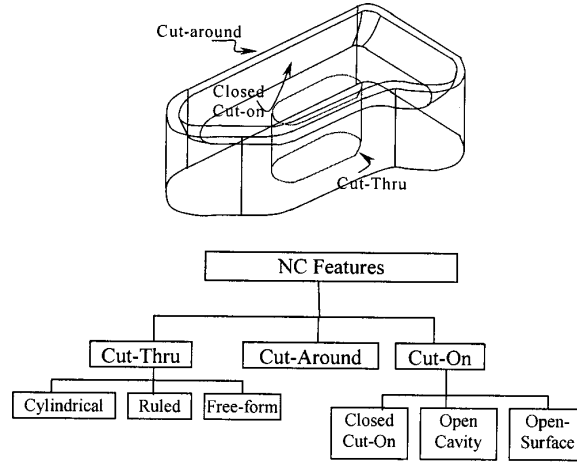


Figure 2.3: CNC machining feature taxonomy according to [159].

Another common observation regarding features is their sensitivity to interactions with other features. It means that FR processes can be easily perturbed when a feature is not precisely matching its definition or, alternatively, a feature definition is often simplified to avoid referring to the very wide diversity of geometric configurations that occur when a given feature interact with many others. It has also to be observed that feature definitions are not available in DMUs (see Section 1.7) partly because of the previous observation and partly due to the fact that features follow definitions targeting very specific applications, which justifies their absence in a DMU since it is regarded as a common repository to feed a large range of product development tasks. As a complement, FR approaches as well as feature modeling ones process strictly standalone components whereas Section 2.3.1 has shown that functional information requires the geometric interaction between components to be precisely characterized. Therefore, functional features must be addressed at the assembly level, which is the purpose works visited in the following section.

2.4.2 Functionality as a result of geometric interactions

The strong ties between geometry and functional semantics are again brought forward by [125] where authors analyze causal kinematic chains of a product based on component-to-component kinematic links deduced from their geometric interfaces.

Authors made simple assumptions about the assembly to semi-automatically infer motion functions:

- Parts having rotational and translational (partial) symmetry properties enable rotational and translational motion respectively, along their

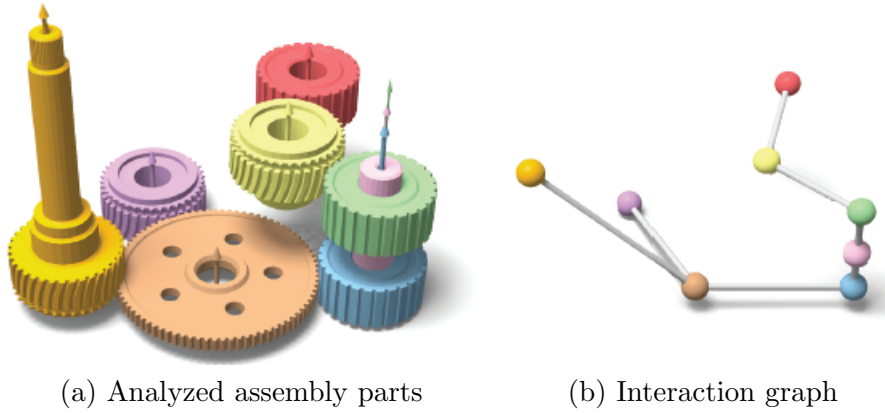


Figure 2.4: Mitra et al. [125]. Parts of an assembly with automatically detected axes of rotation or symmetry (a), and the corresponding interaction graph (b). Graph edges encode the types of joints.

symmetry axis and direction respectively;

- Levers, belts, and chains have few geometric characteristics that enables to identify them automatically: their 1D structure can be analyzed with principal components analysis to infer curvilinear and periodic motion properties;
- Kinematic functions as gears are set manually by the user.

An interaction graph illustrated in Figure 2.4, and representing contacts between product components and their contact characteristic is used to draw conclusions. Alongside the reasoning process, reduced user input is solicited interactively.

Although the work acknowledges exploring components interactions as a great indicator to functional and technical properties of the product, the proposed semi-automatic approach builds on an already meshed model rather than a CAD model, limiting its use to demonstrative kinematic simulations as authors suggest. It should also be noted that the purpose in the above-mentioned work is component animation rather than an effective enrichment of an assembly with functional information.

Dixon & Shah [63] provide an expert system for FR that is both graph- and rule-based. Unlike work presented in Section 2.4.1, emphasis here is put on *assembly feature* recognition, as opposed to *part feature* recognition. Authors define an assembly feature as ‘*an association between two part features that are on different parts*’. The proposed system involves a learning-by-example phase in which a user defines assembly features from existing assembly models. The user interactively provides rules that tie together geometric and algebraic parameters of the defined feature. The user-defined

patterns are then used to extract features from an unseen assembly model where assembly features are to be found. The suggested work uses twist and wrench matrices [181] to define structural and kinematic properties of assembly features.

The work is devoted for application such as reverse engineering and re-engineering of legacy spare parts. Although some of the techniques used inspired our simulation-aware approach, no particular attention has been paid to FE application in authors' work.

The developed system accepts assemblies as B-Rep models. However, it does not account for shape simplifications encountered in industrial DMUs (see Section 1.5.1 and Section 1.7). This is a direct implication of the fact that the system only considers *contact* interaction between parts to generate assembly features. Nonetheless, observation shows that functional interfacing can also be represented, in a simplified manner, as volume intersection, or *interference*. Finally, this learning-based approach does not refer to the behavior concept of an assembly or sub-assembly, which could improve the robustness of this feature-recognition approach and would conform to the well-established dependency between form-behavior-function (see Section 2.3).

Literature studied in this section proved that if any functional information is to be learned, this inquiry should start at the components interaction level. To establish the link between shape and function in the light of the above-mentioned observations, the input geometric model should first be analyzed for candidate geometric interactions. The following section looks at what existing work offers in this regard.

2.5 Geometric analysis to detect interactions

Section 2.3 showed the tight link between shape and function. Function, however, is an interactive phenomenon, satisfied by the inevitable interaction of components in a mechanical system [24, 103]. This means that only shapes in interaction produce functionality. If any functional significance is to be deduced from shapes, geometric interactions are to be addressed first to locate their areas, which can influence their function. In this section, we provide an overview of efforts paid in an attempt to analyze assembly models to look for geometric information that is relevant to our research.

2.5.1 Geometric interaction detection

Geometric interaction detection often drew the attention of researches from different domains, particularly robotics and computer graphics because of its application to collision detection [101, 39, 44, 139].

Lin & Canny [113] provided an efficient technique to determine closest points between two given 3D objects. The algorithm can make use of an

approximate initialization, when available, to converge faster to an accurate solution. This gives it an advantage when considering dynamic environments such as motion planning and robotics, where closest points between pair of objects can be computed adaptively with respect to time [44]. The local convergence nature of this algorithm, however, makes it limited to convex object shapes.

To account for non-convex objects, Quinlan [138] suggested the division of the object into sub-components, which in turn are convex themselves. The problem of finding the closest points, thus the minimal distance, between two non-convex objects reduces to finding closest points between their sub-components, then considering the pair with minimum distance. To reduce the quadratic complexity inherent to this approach, the author suggests using cheap bounding spheres checks to reduce the number of compared components.

Works from Agrawala et al. [20] and Mitra et al. [125] have built on minimal distance detection to efficiently determine geometric interactions in an assembly, such as contacts and clearances. However, all of the above-mentioned works considered a discrete geometric model of type polyhedral, a representation that is not commonly used in industrial DMUs (see Section 1.5) and not robust enough to correctly detect unambiguous interactions between components in complex assemblies. This is exemplified in commercial CAD software like CATIA V5, where interaction detection is simply displayed and left to user's interpretation but cannot be used for further assembly geometry processing.

This inconvenience was addressed, and tackled in the work of Iacob et al. [89], where a contact detection algorithm is provided based on analytical Non-Uniform Rational B-Spline (NURBS) surfaces. Detected contacts between components in a DMU are then used for assembly/disassembly planning, and in VR application. Due to its particular application, this method paid no effort to detect interaction zones. Such a geometric knowledge, however, is a key element for processing DMU for FEA purposes.

The recent work of Jourdes et al. [95], in the framework of ROMMA project [1], solves the problem of the detection of precise contact zones in a highly efficient manner, making use of discretized techniques that exploit the Graphics Processing Unit (GPU) computational power. Despite its use of internal discrete models to communicate with the GPU, the proposed algorithm still inputs, and efficiently produces, analytical NURBS surfaces. This work, however, did not address interference zones detection.

2.5.2 Importance of a unique geometric representation

In the domain of shape recognition, certain criteria have been identified to ensure relevant comparisons between shape descriptions. One property of shape synthesis methods has been outlined that allow shape search and 3D

pattern detection. This is *representation uniqueness* [122, 126, 90].

Uniqueness implies a one-to-one relationship between a shape and its descriptor using a given representation [90]. This means that using an appropriate representation, an object can be geometrically described in only one way. Two different descriptions mean that the underlying objects are geometrically distinct. Uniqueness in this sense also implies *invariance* [126], that is if two objects have the same shape, they must have the same geometric description under their descriptor.

Uniqueness is a mandatory property to enable the judgment of whether or not two descriptors represent the same shape. Work as early as [122] shed light on this requirement when retrieval of relevant information out of a geometric model is considered.

Commercial CAD systems, however, do not account for shape uniqueness. In fact, a given object, such as a simple cylinder, may be represented through different ways and different number of faces and edges even when using the same geometric modeler, as Section 5.2.2 shows. In order to enable a robust geometric interaction analysis, this inconvenience is to be addressed first, which is the topic of Section 5.2.2.

Today's industries regard the DMU as the product knowledge repository. They expect it to tie the product geometric model with related information, such as functionality, in a formally structured manner. Works in this direction join our endeavor to enrich the dmU with functional knowledge. Similar efforts are thus summarized in the upcoming section.

2.6 CAD and knowledge representation

CAD systems are meant to provide a PDP with tools that spare the designers the burden of repetitive tasks, allowing them to concentrate on creativity and core expertise. They are the evolution of drafting tools, that use ever-growing computing capacities and interactive techniques.

Design activities are becoming more and more knowledge-greedy. The availability of relevant information is taking a major part in an efficient PDP. Designers reportedly spend up to 60% of their production time searching for the right information [108]. Ullman [171] argues that knowledge reuse is involved in more than 75% of design activities.

CAD systems are, thus, more and more required to equip designers with needed engineering knowledge. However, observations show that this knowledge is still scattered around the DMU in a non-structured manner. Most of this knowledge comes in a free text format (see Section 1.7), which is neither reliable nor robust.

Section 2.6.2 studies research that tackled this problem by means of general knowledge management tools applying paradigms of the Semantic Web. Section 2.6.3 looks through prior work that utilized an approach

more specific to engineering knowledge. We first make a distinction between knowledge at the domain of discourse level, and knowledge at the current instance level in Section 2.6.1.

2.6.1 Domain knowledge and model knowledge

In fact, shortly after CAD systems were introduced and commercialized they were suggested to provide active feedback to the designer, to enhance the engineering process with decision support systems, embedded in their working environment. Those systems made use of engineering knowledge about both the domain and the particular product under development.

In the context of a PDP, we identify knowledge about the underlying domain, such as aerospace and automotive industries, or software development, which we refer to as *domain knowledge*. Another type of knowledge that can be distinguished is the knowledge about a particular product or instance of this product, e.g., car engine, aircraft, or piece of software. We refer to such knowledge as *model knowledge*. This distinction is purely pragmatic since it allows domain knowledge to represent global expertise independently from any information about a particular case. Model knowledge, however, only makes sense in the context of domain knowledge.

2.6.2 Ontologies as an assembly knowledge storehouse

The concept of ontologies as it applies to computer science is closely related to the Semantic Web [35]. The Semantic Web is seen by World Wide Web Consortium (W3C) as the Web of data, as compared to the Web of documents that we know. It enable machines to interact and connect to each other in the same way human beings do through the Web. To this end, a common machine interpretable language, or vocabulary, should exist. Ontologies are the Semantic Web vocabulary [5]. At their simplest understanding, they define concepts and relationships between them in a machine understandable language.

Because of their established formalism and their ability to intuitively model the facts we know about a given domain of discourse, ontologies are widely used as knowledge repositories. For a particular domain, knowledge is represented as a set of objects, referred to as *individuals*, that are grouped in classes that are called *concepts*. Classes are organized in a hierarchical manner to reflect the generalization relationships between sets of objects. Individuals are connected with relationships that are called *roles* in the context of ontologies. Individuals, concepts and roles are identified by means of agreed-upon human readable vocabularies. Gruber [76] describes an ontology as a commitment to these vocabularies between participants to an Artificial Intelligence (AI) system.

Knowledge captured by an ontology is classified in two categories:

1. The terminological box, or *TBox*, where general concepts and rules are expressed. This typically maps to the domain knowledge;
2. The assertional box, or *ABox*, where information about instances and their relationships are maintained. The ABox typically maps to the model knowledge.

It is important to note that while ontologies formally define the common language necessary for knowledge sharing, they leave the choice open for how this language is represented and communicated. Gruber [76] distinguishes three needs to allow knowledge sharing:

- A common representation language format;
- A common agent communication protocol;
- And a common specification of the content of shared knowledge.

An ontology fulfills the last requirement, while the first two items are considered to be implementation details rather than conceptualization problems. The use of an ontology in the Semantic Web compares to the use of a given language, e.g., English, French, or Arabic, in the Web where vocabulary and their semantics are well defined and understood by people speaking the language.

A variety of solutions already exist to represent ontologies in a common format. Names include Resource Description Framework Scheme (RDF-S), Ontology Interchange Language (OIL) and Web Ontology Language (OWL) family [14] including variants like OWL Lite, OWL DL and OWL Full. This compares to the use of HTML in the Web to represent documents. Protocols do also exist to allow exchange of facts and queries using ontologies, examples are SPARQL Protocol and RDF Query Language (SPARQL), DL Implementation Group (DIG) [15] and Simple Semantic Web Architecture and Protocol (SSWAP). This compares to the use of HTTP to communicate requests and responses on the Web.

Ontologies are often used in different engineering disciplines to capture knowledge. Liang & Paredis [112] provide a *port ontology* as an unambiguous semantic structure that combines form, function and behavior as design information characterizing subsystems interactions in a given mechatronic product. This ontology, however, makes no connection between these three design aspects. In Rahmani & Thompson [142], the authors build upon the previous work and show how to represent functional interfaces between product subsystems in machine-interpretable manner using a three-layered ontology (two layers for domain knowledge and one layer of model knowledge). They also provide necessary means to verify functional compatibility between system components through their functional interfaces, thereby

called ports. In fact, this work is general and applies to different disciplines beside mechanical engineering.

These two works join and extend a heritage of literature in an effort to aid the design process in providing relevant information in a globally understood basis.

Kitamura & Mizoguchi [104] suggest a semantic framework to enable conceptual engineering knowledge sharing about functionality. This framework is implemented in terms of layered ontologies where concepts of each layer builds upon those of the layer above. To guarantee the generality and wide coverage of their approach, the authors emphasize the distinction between what a function is and the way it is achieved. This is reflected in a distinctive conceptualization in two separate layers; *functional concept ontology* and *functional way knowledge*, respectively. For example, ‘welding’ is more than a function, under the proposed framework, as it implies ‘fusion’ as a way of satisfying the function ‘unification’. This function, however, can be satisfied by other means such as ‘fastening’. Authors promote their ontological framework as an agreement about a common vocabulary to allow designers and engineers to share knowledge.

In the continuity of their work [103] presented in Section 2.3.3, Kim et al. [102] developed the *AsD ontology* to capture design intentions in a heterogeneous collaborative assembly design environment. Authors do not only use advances in the domain of knowledge representation to formally represent ARM presented in [103] using ontologies, they also use inferences to obtain new facts that are not implicitly available in the initial model. Inferred facts, however, dwelt in the domain of consistency checks, joins types, and relative DoF. Figure 2.5 shows an overall structure of the proposed system and its connections to other modules to deliver assistance to engineers during the assembly design process. This figure shows how an inference engine is used to extract implicit knowledge, then assistance is provided mainly through querying, using a semantic search engine.

All the previously cited approaches share a common perspective since they address the design assistance in a top-down manner where functional information has to be related to 3D component models or to their interfaces by an engineer. Consequently, this information may be incomplete if an engineer fails to perform some connections during the design process. Few authors have taken advantage of inference mechanisms to automate the connections or check the consistency of the overall design data. Indeed, setting up connections between component geometry and functional informations raises the question of the meaning of this connection. It is essentially a logical connection between data, e.g., functional ones, and an instance of 3D component and it is not clear whether this connection should target a subset of the component and what should be the appropriate geometric entities. If a designer had to query this functional model to highlight a geometric area over the boundaries of components, these approaches cannot process such

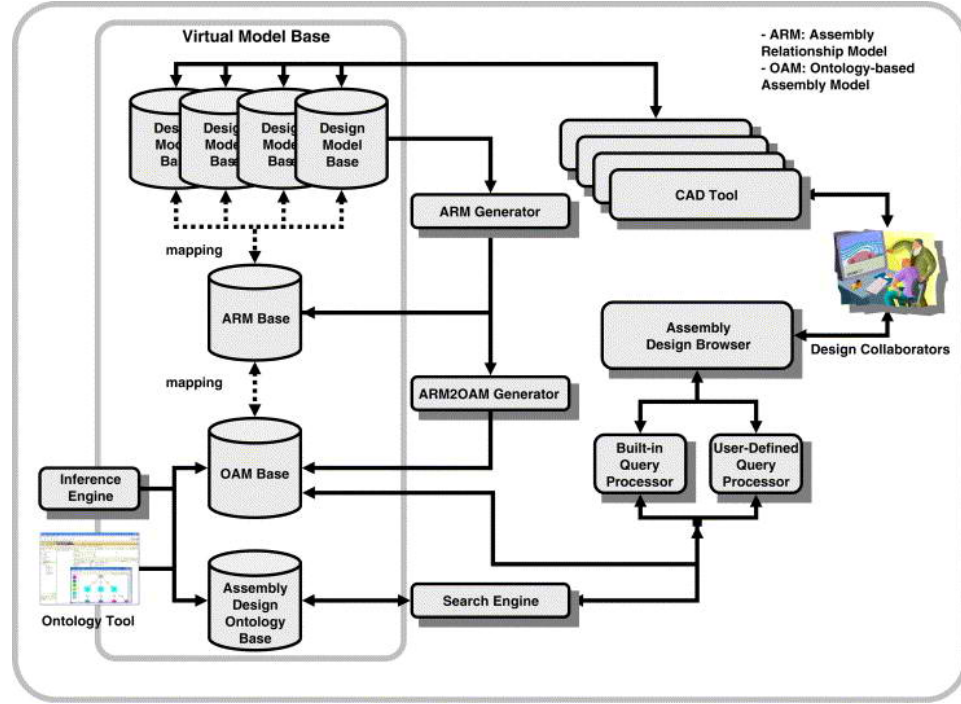


Figure 2.5: Assembly design information sharing framework as proposed by Kim et al. [102].

query, showing that the proposed connections are not adequate to process 3D components for FEA preparation. Combining this observation with the top-down feature of these approaches we infer that processing a DMU containing essentially geometric models of components (see Section 1.6 and Section 1.7) in a bottom-up manner, cannot be automated with these approaches because the connections between geometric and functional informations are missing.

Barbau et al. [32] cover a large spectrum of product description with an ontology, referred to as *OntoSTEP*, incorporating both geometry and structure of a DMU as available through STEP APs [8, 12], the Core Product Model (CPM) introduced in [67] and the Open Assembly Model (OAM) introduced in [34]. A tool was developed to translate an EXPRESS scheme [13], the scheme that governs STEP files syntax, into an OWL DL ontology, defining its TBox. A STEP file can then be imported into the same ontology, using the same tool, to define an ABox. The tool was implemented in terms of a plug-in to the ontology editor Protégé [3]. Authors thus define a mapping between EXPRESS primitives (entities, instances and attributes) and those of OWL (classes, individuals and properties, respectively) to enable the import of a TBox. They also implement a syntactic

analyzer that parses the STEP file to create the corresponding ABox of a given model. The work aims at establishing a semantic layer on top of an EXPRESS description. Aligning these semantics with functions and design intents expressed in models such as CPM and OAM allows reasoning and extraction of implicit knowledge. The authors use Description Logic (DL) reasoner Pellet [156]. In spite of a relative success, authors showed that not all EXPRESS language constructs can be expressed using OWL DL. Examples are functions, entity constraints and attribute calculations that may require complex algorithms beyond the expressiveness of DL, the logic upon which OWL DL is built. Authors conclude that not all aspects of STEP can be rigorously reflected through ontologies alone, leading to limitations on reasoning power.

These approaches are meant to accompany the design process, and to lend it the necessary tools for modeling and verification, including means to represent knowledge about system components interactions as major actors to such a process. Some of the work analyzed provided means to extract technical or functional implied intention from an existing model through inferences. This information, however, did not go further than interface-wise descriptions of assembly links and kinematic connections. Inferences are also used to align geometric models to functional ones. However, the latter were explicitly provided and reasoners were rarely used to merge knowledge stemming from different models. Even as knowledge capturers, proposed ontologies fail short to encapsulate functional knowledge thoroughly enough to the point that satisfies FEA requirements (see Section 1.9) with precise geometric information about interfaces between components as well as functional information about these areas.

2.6.3 Knowledge-based engineering approaches

Engineering knowledge is spread out in different places and forms along a PDP. This includes experts' minds, worksheets, CAD models, company codes, databases, flowcharts, implicit and explicit conventions and rules of thumb, etc. Knowledge Based Engineering (KBE) aims at gathering all such knowledge in one place, and make it accessible to actors of the PDP at any stage.

KBE can be seen as a specific type of experts system as applied to engineering field. They combine geometric modeling, configuration management, and knowledge management into one rule-based system [115].

Domain knowledge is collected and stored into a knowledge management module, then rules derived from this knowledge are applied to CAD models in a parametric-modeling-like manner. This knowledge also governs other engineering and manufacturing aspects rather than geometry through the configuration management module.

In this sense, KBE extends traditional CAD systems. DMUs are enriched

with information that is persistent, relevant, meaningful, and reusable. This knowledge is then analyzed and used to provide the designer with decision making tools and advisory modules. This is meant to save development time, and allow a designer to focus on innovative aspects rather than mundane tasks.

The way knowledge is organized in a KBE system enables also the re-usability of pre-existing components or sub-systems, remarkably reducing the cost of products modifications or upgrade. This is particularly useful in industries where the design activity is of an adaptive nature; that is products are rarely redesigned from scratch, but models are often adapted to emerging needs. In this case, KBE aims at capturing the design intent in the model itself, allowing for easier modification and avoiding reverse engineering efforts to guess what engineers had in mind when the model was first created [37].

KBE has its roots back in the late 80s [37, 176]. Many successful applications reaped its fruits in the beginning of the century. Chapman & Pinfold [48] show one application in the domain of automotive industry, applying a standard KBE system in a highly dynamic design environment. In the aerospace industry, La Rocca & van Tooren [145] tell a success story fitting KBE to multi-disciplinary design to enable automatic generation of FE analysis models. Emberey et al. show another application of KBE in the domain of aeronautics [65].

Considering KBE early high potentials, it seems that this approach didn't yet meet its expectations, despite numerous success stories. This led people to rethink the utility of such investment. Others tried to criticize KBE, studying both failure and success case-studies, and drawing conclusions about where did the applications of KBE go wrong [176].

One argument about the shortcomings of KBE is the lack of explicit methodologies. Although such frameworks exist (MOKA [161], KOMPRESSA [115], KNOMAD [56] and DEE [144]), applications usually don't commit themselves to any, and tend to be case-based. Verhagen et al. [176] note that more than 80% of KBE applications do not fit in a particular framework, nor do they follow any well-defined methodology. This poor modeling contradicts with KBE basic assumptions and leads to significant loss of knowledge.

Another problem is the lack of a semantic link between identified formulae, rules and models on one side, and real-life engineering expertise and understandings on another. This reduces collected knowledge to mere data that still miss the context of application. This contextual gap appears at the level of knowledge collection, as well as knowledge representation. Knowledge representation models are still unable to capture the link between one engineering element and its scope of validity. This shortcoming strongly hinder re-usability, one of major advantages of KBE, making initial overhead of KBE systems unnecessarily costly.

The lack of quantitative means to assess the 'success' of a KBE system

is another major inconvenience. A KBE implementation should be compared to its traditional counterpart to truly justify the use of such initially costly approach. Although some work shows serious comparative studies to advocate KBE use [65, 55], this doesn't apply for the majority of related work [176].

Quantitative measures are also strongly needed to assess the suitability of KBE to a particular project or application. A primary reason that made people drift away from investing in KBE is the failure of inadequate applications that were motivated by early success of such technology. In such situations, extra-cost was often not justified by the little gain, because of the nature of the application in hand [37].

Looking more precisely at the reasons that restrict the extensive use of KBE systems, the connections between technological parameters and geometric entities is similar to the connections observed in Section 2.6.2. Consequently, the connections set up are applicable to a fairly small set of configurations, i.e., when shape modifications are performed the new configurations are no longer compatible with the technological parameters they are connected to. This happens because these connections address geometric areas of 3D components that are not precisely reflecting the meaning of their associated technological parameters.

The need of robust connections between geometric elements, down to the level of geometric interaction zones, and functional knowledge in terms of agreed-upon semantics is emphasized when considering application such as FEA. The following section walks through recent approaches in this domain.

2.7 From CAD to FEA

Section 1.9 introduced the finite element method, and how it contributes to the whole PDP. In this section we analyze efforts paid to automate or semi-automate the generation of the FEM.

2.7.1 Pre-processing at the core of the FEM

Haghighi & Kang [79] describe pre-processing as the most time-consuming and expertise-intensive task in the behavioral simulation process. They also argue that expertise and knowledge invested at this stage have a direct implication on the accuracy of analysis results. The error-prone and resource-intensive nature of this task often makes it the bottle-neck in the PDP. Jones et al. [92] attribute the high cost of preprocessing to the many non-trivial subtasks it involves, such as geometry processing, mesh quality control, and the assignment of physical properties to mesh elements.

2.7.2 Direct geometric approaches

As pointed out by Thakur et al. [166], most of prior work related to the geometric transformations applied to components to generate a FE model from a B-Rep CAD model, are purely geometric transformations, i.e., the component shape is modified using criteria of morphological type. In Makem et al. [117], a component is subjected to a segmentation process into manifold models to enable the generation of a semi-structured mesh is proposed. The hybrid mesh of structured and unstructured zones allows for efficient anisotropic structural simulations. Few contributions take into account FE sizes to modulate the geometric transformations applied [109]. Quadros et al. [137] use de-featuring techniques to reduce model complexity. They therefore generate an intermediate discrete model, keeping backward links to the original CAD model to allow information flow such as boundary condition attributes.

In the present context, the focus is placed on assemblies as a representation of a DMU. There, few research works have addressed the FEA preparation of assemblies. Specific operators have been provided to compute contact zones between components. These operators fall into two categories depending on whether the geometric model used to describe the components is an analytical B-Rep model or a discretized mesh.

In case of B-Rep CAD models, Clark et al. [52] have developed a specific operator to compute the imprint of a component onto another one and use the corresponding imprint to subdivide the boundary of each component so that they share a common geometric area that reflects the contact area between the components. As a result, this common area can be used to generate conformal FE meshes in this area, which greatly improves the FE mesh generation process of an assembly model. In case of meshed or faceted CAD models, several approaches have been proposed [50, 114] to compute the common areas between components representing their contact areas and to process FE meshes generated on a component basis so that their common area can be identified and their FE meshes in that region can be modified to produce a conformal mesh. Obviously, these approaches are of great interest to process assembly models for FEA. However, those referring to a faceted model are not robust since the operators are rather sensitive to discretized representation they use as input. More precisely, it becomes difficult to make a distinction between a discretized representation of two cylinders in contact with each other along a cylindrical surface and a discretized representation of two interfering cylinders as it can happen in a DMU with screws and nuts. For this reason, this approach is not suited in the present context. Regarding Clark's approach, it has been addressed using Boolean type operators inside a specific CAD software and it is restricted to contact configurations. As a result, it is not generic compared to Jourdes's [95] approach that uses a STEP file as input and projection-type operators that can adapt to accuracy

of relative position of each component.

All these approaches, however, did not show any connection to the functional attributes of a geometric interaction between components, leaving an open question of how adequate those adaptations are with respect to a given simulation process under prescribed simulation objectives. In the context of the ROMMA [1] project, work has focused on assembly processing for FEA and has highlighted the need to specifically process the interfaces between components since they are directly related to the simulation objectives.

Work studied in the context of CAD-to-FEA transformations showed that assembly models are rarely considered as a whole, instead, components models are processed and transferred between the two domains individually. Methods that accounted for interaction between components are also uncommon in the literature, and those who did, only considered geometric contacts as functional interaction indicators, leaving prevailing industrial conventions, such as volumetric interferences, uninterpreted. This is a natural consequence of the shortage we pointed out in Section 2.4. This shortage reflects the need for a concrete approach that translates product geometry to simulation-relevant functional properties, while taking into account mainstream industrial practices, and the integrity of an assembly model.

2.8 Conclusions

In this chapter, concepts such as functionality and the relationship between function, behavior and form are considered from literature standpoint. These three concepts are inter-related, which means that adding functional information to a purely geometric model of a system should refer, somehow, to the behavior of this system. In addition to these three concepts, the concept of state, though not mentioned in the relationships between function, behavior and form, can be related to function. In later chapters, these concepts will be revisited, extended, or narrowed down to a more specific context or definition.

Examining work that compares to ours, in terms of problem tackled, showed that though it is possible to recognize some basic manufacturing features by merely considering local geometric properties of components, the detection of more complicated functional properties, such as these required for simulation preprocessing, requires that the geometric model be regarded from a wider angle, that also covers the interaction between different components. In addition, the feature recognition approaches essentially concentrate on standalone components which prevent them from addressing functional issues since their interfaces with other components are not taken into account.

Knowledge repositories and reasoning methods used in the context of a DMU are also examined, to determine that, although ontologies showed

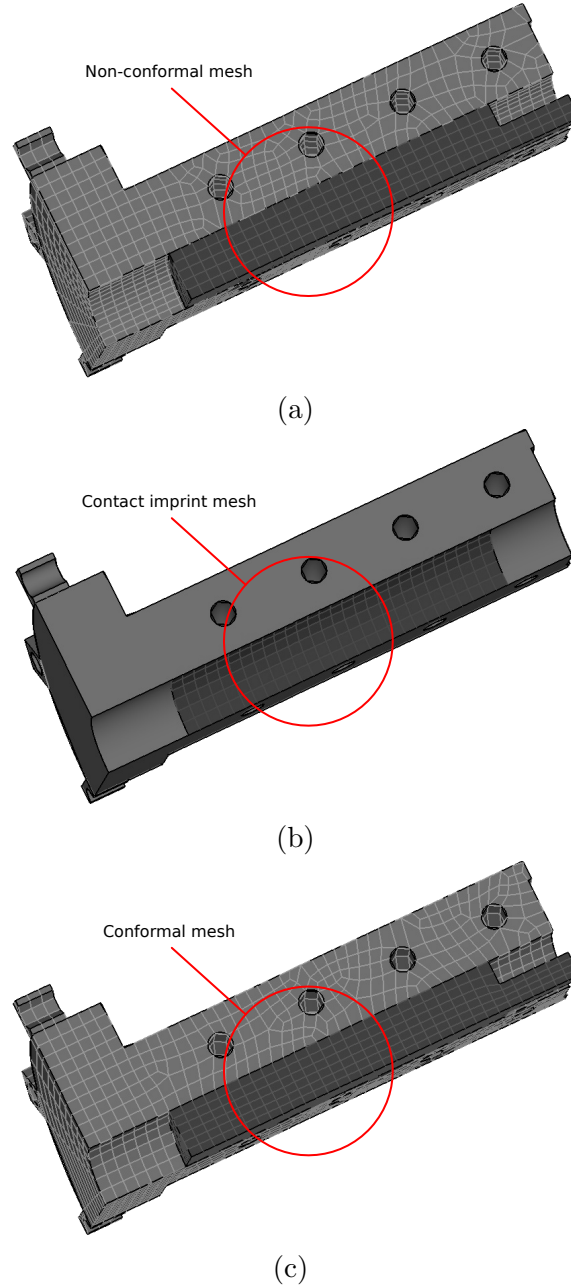


Figure 2.6: A demonstration of the work of Clark et al. [52]. (a) A non-conformal mesh of an assembly of two components. (b) Components imprint onto the each other represents the contact zone, a shared mesh is generated at this region. (c) A conformal mesh of the same assembly, where the mesh of the rest of each volume is generated after the mesh of the shared surface (the imprint).

promising abilities to faithfully represent engineering knowledge, and to reason upon it to a certain extent, they are still inadequate for the type of reasoning that requires heavy calculations or complex algorithms, as it is the case in 3D geometry processing. This is a strong requirement to be able to process complex industrial assemblies.

In spite of its potentials, little efforts have been paid to exploit the Semantic Web reasoning capabilities to extract new functional knowledge from merely geometric one to the level where the former can be used in the preparation of a model for FEA purposes. Alternatively, the use of engineering-specific approaches such as KBE enables flexible adaption to reasoning needs. However, such approaches still miss the semantic connection to design rationale, and come at a yet unjustified high cost with a very limited capability to adapt to product variants and even more to an acceptable range of products. At the origin of this limitation stands the structure of the geometric model supporting the KBE application and how it is connected to knowledge representation.

Attempts to automate the preprocessing of a FEA showed that only few works made the necessary connection to the functional properties of components and their interfaces. These approaches are still needed for automated function enrichment of DMUs and must produce an accurate geometric representation of the interface areas between components to be useful for FEA preparation.

Chapter 3

From Geometry to Functional Semantics: Needs and Objectives

This chapter sets the objectives of our work, shedding the light on the prominence of functional knowledge, and on the importance of its inference at different levels of the DMU structure, while minimizing user's interactions. We also show the applications and implications such an inference have on the acceleration and enhancement of a PDP, particularly in the context of FEA preparation.

This chapter is organized as follows: Section 3.1 presents the need for automatic and intelligent preparation tools to adapt DMU data to FE simulations, Section 3.2 shows that geometric assumptions are made about the DMU, reflecting a variety of industrial conventions. These conventions are to be taken into account if the shape of a DMU is to be interpreted for FE simulation applications. In Section 3.3, efficient methods for timely preparations of simulation models are shown to require an enrichment of the DMU content to incorporate critical functional knowledge, while preserving connections between functional and geometric entities. Section 3.4 enumerate three different levels at which this functional enrichment of a DMU content must take place, namely, the component interface level, the component level, and the group of components level.

3.1 Taking 3D models beyond manufacturing purposes

With the development of 3D modeling techniques, industrial blueprints, their 2D counterparts, have become less prevalent across the production

process. This technical leap enabled the utilization of a reference model, now referred to as a DMU, at different stages of a PDP, especially as an entry point to simulation processes (see Section 1.4.2). However, this advent also came at a cost: technological and functional information are scattered around the DMU in a disorganized manner (see Section 1.7), the abstraction of assembly geometry stopped being standardized (see Section 1.5.1) and thus, became unreliable, unlike the way it used to be with 2D technical drawings.

This technological and functional knowledge remains a core requirement for the use of a DMU in product development tasks such as finite element analyses and simulations [42]. Huge manual efforts are being paid by engineers on daily basis to reconstitute such information [108].

Geometric modelers, as part of CAD software products, provide tools for intuitive authoring and manipulation of DMUs. These substantial advantages over two-dimensional drawings provoked a tremendous change in the field of geometric modeling. As a result, traditional blueprints gave way to 3D models in today's design offices. In this section, we outline the potential that a DMU has to actively participate to the leverage of the PDP.

As a central component to a PDP, its DMUs allows engineers to design components shapes and position them before the product is actually manufactured and put to operation. Section 1.4 explains the focal role the DMU plays in a PDP with the increasing tendency in today's industries to relate different tasks to the DMU content, all along the PDP. Reciprocally, there is also a tendency to adapt the DMU content to the PDP requirements. A DMU shows its capacity to contain further information rather than pure geometry. Examples are component materials and their properties, kinematic connections between components, geometric constraints and functional zones to name only few (see Section 1.7).

Considering this viewpoint, a DMU can do better than barely providing references for manufacturing, since it is, indeed, communicated all across a PDP. One can expect DMUs to serve as entry points for simulation purposes, e.g., allowing the generation of digital product prototypes. It would be convenient if the same model, i.e. the DMU, could be enriched with necessary knowledge for subsequent stages of the PDP, that would definitely accelerate preparation processes, e.g. FEA ones and others [64].

Nonetheless, DMUs, the way they are designed, are subjected to manufacturing requirements. This is mainly because designers oftentimes have this consideration in mind while they're also bounded by the solid modeler capabilities when creating 3D models [102]. For this reason, the DMU is not promptly ready to play its polymorphic role in the PDP, in a reference to this concept set at Section 1.10. In fact, to process a DMU for FEA, geometric transformations are still required to adapt it to simulation requirements. Section 2.7.1 pointed out that this high skill demanding task still poses a problem to the efficiency of a PDP, and that automating it as

much as possible gets in the way of timely simulations.

Most of today's vendors ship their industrial CAD systems with modules such as kinematics simulation, FE simulation for structural, thermal, and flow analyses. These modules provide tools for additional tasks of a PDP rather than mere geometric modeling during the design process. However, the lack of automated and *intelligent* adaptation methods hinders the utilization of these FE analysis modules.

Some sort of intelligence is thus required in order to adapt a DMU for development phases other than manufacturing along a PDP. Section 2.7 however, showed the shortage in the state of the art of an robust approach that functionally interprets the geometry of a DMU as an assembly of interacting components, down to the level of interacting surfaces, while taking into account dominant industrial conventional representation of such interactions (see Sections 1.5.1 and 1.7). Such an approach would faithfully bridge the gap between CAD offerings and FEA needs.

A major objective of the proposed approach stands in exploiting the DMU content for simulation purposes. This content must be processed in a bottom-up manner since a DMU content reduces to robust information only for mere geometric models of components. Section 2.6 has shown that top-down approaches don't bring a tight connection between 3D models and the technological, functional data associated to components and assemblies, and they have not emerged in commercial CAD systems. More precisely, this objective addresses the generation of a simulation model (as introduced in Section 1.9.2) that is suitable for timely and accurate FEA under prescribed simulation objectives.

3.2 Differences between digital and real shapes

The geometric model of each component in a DMU is meant to provide a precise product model to enable its manufacture, as mentioned earlier in Section 1.4. Inaccurate digital models are therefore little tolerated, as such inaccuracy puts the manufacturing process at stake. However, particular geometric configurations, e.g. helical threads and involute gear profile, are not used as input of manufacturing processes. Also, modeling such surfaces and volumes as carbon copies of real shapes is a tedious and inefficient task that is of little or no interest to the PDP. In fact, the geometric accuracy of such features has no impact on the manufacturing process of the components due to at least one of the following reasons:

- Components such as threaded bolts, nuts, and profiled gears are often imported as third-party components [4] that comply to specific standards¹;

¹Naturally, precise detailed geometry of complex surfaces such as threads and gear teeth on molded plastic components may be however important when manufactured in-house.

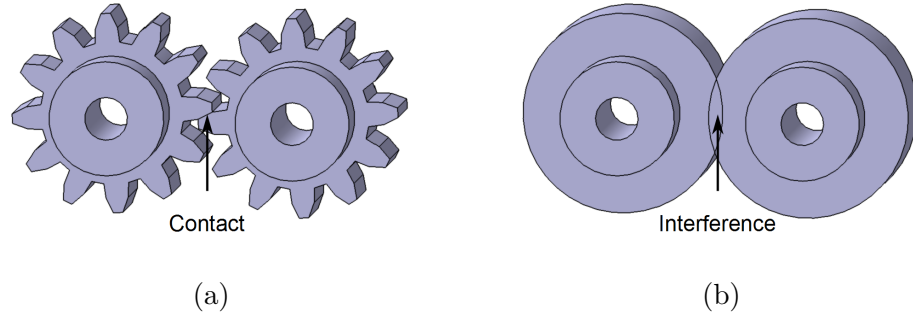


Figure 3.1: Two engaged spur gears: (a) representation of real surfaces, showing involute profiles, and a simple contact between two gears; (b) simplified representation as simple cylinders, leading to an unrealistic interference.

- The machining of profiled gears, threads, spline profile, etc. can be a generative process that is prescribed by tooling parameters set later in a PDP than the design stage. Consequently, at design and simulation stages in a PDP, these shapes need not be accurate.

Other examples can be easily observed in industrial DMUs since these differences between digital and real shapes is current practice.

Consequently, and for the sake of efficiency, complicated surfaces that are part of imported components, or that comply to predefined implicit or explicit standards are often simplified. For instance, threads and gear profiles are modeled as simple cylindric surfaces as shows Figure 3.1. This simplifies the geometric modeling task, while preserving technical informations for manufacturing.

Those simplifications, however, imply an interpretation from the engineers. For instance, both a brake disc and a gear may be represented as a simple cylinder after geometric simplification. Hence, the mechanical component has to be studied in its environment to clarify its nature, i.e., the component must be analyzed along with its interaction with neighboring components (see Figure 3.1). It is also worth observing that, as a consequence of these simplifications, the geometric interactions between neighboring digital components is not limited to contacts or clearances, as it is the case between real components. Indeed, digital models of components may exhibit volume interference (see Figure 3.1) while still conventionally representing a consistent configuration of their real counterpart.

In addition to geometric simplifications, another inconvenience about geometric models of a DMU is the way geometric interactions are handled. We have seen in Section 1.6.2 that geometric constraints such as contact and coaxiality may be deliberately dropped and replaced by absolute positioning,

for the sake of conciseness or re-usability. Even when positioning constraints are kept, they are still incomplete to infer geometric interactions between components, as previously established in Section 1.6.2. For instance, the simple fact that two cylindric surfaces are coaxial does not necessarily mean that corresponding faces are in contact, at play, or having an interference.

These shape differences and representation shortcomings render the judgment about functional intentions of elements of a DMU a non-trivial task, even to a knowledgeable eye. This implies the incorporation of different industrial conventions into the knowledge base of an expert system, if any meaningful functional information is to be extracted. It is a second objective of the proposed approach to structure the knowledge related to component and assembly representation so that it can be processed reliably and effectively connected with 3D shapes of components and assemblies.

3.3 Enabling semi-automatic pre-processing

To speed up a PDP, aeronautical, automotive and other industries face increasing needs in setting up timely FE simulations of large sub-structures of their products. The challenge is not only to study standalone components but also to simulate the structural behavior of large assemblies containing up to thousands of components [1, 41]. DMUs are widely used during a PDP as the virtual geometric product model (see Section 1.4.2). This model contains a detailed 3D representation of the whole product structure available for simulation purposes. To prepare large sub-structure models for simulation (such as wings or aircraft fuselage structures); the DMU offers a complete geometric model as an input (even though not necessarily a faithful one, as seen in Section 3.2). However, speeding up the simulation model generation (see Figure 1.19) strongly relies on reducing the time required to perform the geometric transformations needed to adapt the DMU to FE requirements in the context of the pre-processing step discussed in Section 1.9.2.

3.3.1 Pre-processing tasks

Currently, due to the need of geometric transformations required to adapt the shape of a DMU to simulation objectives (see Section 1.9), the corresponding adaption of CAD models to generate FE models still requires time and specific skills because there is a lack of automation of these transformations. The time required to generate FE models often prevents engineers from using structural analyses during early stages of a PDP. Several authors proposed approaches to automate shape transformations required for a standalone component (see Section 2.7). However, very few research work addresses assembly models where similar configurations are duplicated many times, e.g., contact areas, bolted assembly joint FE models. Consequently,

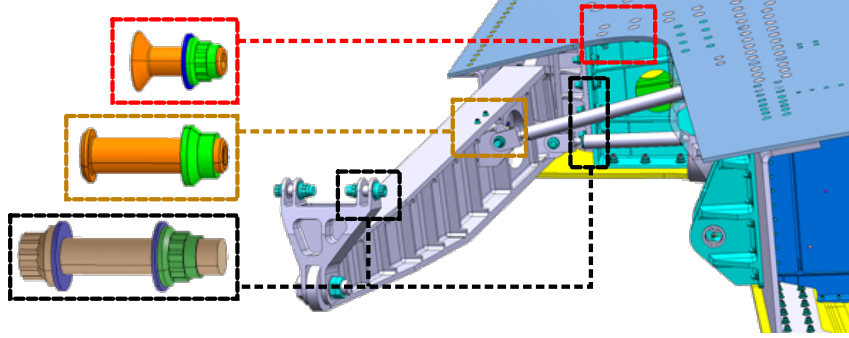


Figure 3.2: Examples of aeronautical DMUs with a variety of bolted junctions [42].

DMU of aeronautical structures are particularly complex to transform due to their large number of joints incorporating bolts or rivets (see Figure 3.2).

Domain decomposition and shape transformations mentioned in Section 1.9.2 are examples of interactive and error-prone processes that an engineer must perform tediously to enable efficient FE simulations. Within the available resources and time frames planned in an industrial PDP, engineers are bounded to simulate small models rather than complete assembly structures. It is an objective of the proposed approach to contribute to speed up and automate the shape transformations of assembly models for FE simulations.

3.3.2 Pre-processing automation requirements

It can be observed that repetitive tasks originate from similar configurations like bolted junctions and, more generally, interfaces between components. Similar tasks relate to shape similarities as well as behavioral similarities since the shape transformations performed fit into the same simulations objectives for a given FE simulation model. As pointed out in Section 2.3, shape, behavior and function are independent concepts and shape and behavior similarities can refer to function similarity, i.e., similar shapes behaving similarly and contribute to similar functions. Indeed, it is the case when referring to bolted junctions where the underlying function is the ‘assembly of components using bolts’. This analysis shows that functions are good candidates to complement shapes when they have associated interfaces, whereas feature recognition, purely based on component geometry, do not enable a direct connection to component function (see Section 2.4).

Here, the targeted FEA preparation aims at producing a quantitative behavioral analysis of an assembly, e.g., the computation of stress, strain, and displacement fields. Therefore, there is no such behavioral information

available to combine with shape information that can help derive functional information about components in addition to their shape. However, if there is no quantitative behavioral information available for components, it is possible to refer to the design rationale where design solutions emerge from a qualitative assessment of components at the early design stages. Similarly, qualitative behavior assessment is common engineers' practice when analyzing a mechanism from either blueprints or DMUs.

To this end, the above-mentioned principle is a path to another major objective of the proposed approach to automate FEA preparation processes. Indications about components functions, functional groups, and functional interactions can be gained from assembly geometry processing and behavioral information. These indications must be coupled with the product geometry, not only at the component and group of components level, but also at the joints interface level, particularly in connection with functional interactions. This may imply a *functional* restructuring of components geometry to highlight interaction zones. The following section sheds more light on this issue.

3.4 Bridging the gap with functional knowledge

Despite attempts of geometric modelers vendors, as well as efforts paid by data exchange standardization committees, industrial practices in the field of knowledge representation and communication are still far from being standardized, as shown in Sections 1.7 and 2.6.

3D modelers still fall short of providing a unified method to maintain technical and functional properties alongside geometric models of components and assemblies. Figure 1.2 shows precise technical annotations of dimensioning and tolerancing which are standardized for a shaft-housing connection (see Appendix A). Nevertheless, in a platform-independent 3D representation of a product, such as a STEP file [7], this knowledge is lost as both shaft and housing are represented with their nominal diameter, with no further information about dimensional tolerances. Figure 3.3 depicts an example of a 3D model showing a piston fit in a cylinder sleeve. This fit should be loose in order for the crank-piston mechanism to work properly. However, both parts of the fit are represented with the same nominal diameter, leaving the fit nature ambiguous.

Even when standards provide auxiliary annotations that may actually hold functional information (as Section 1.7 showed), observations reveal that current CAD modelers do not make use of these facilities, stripping their native models of all information but mere geometry when exporting them in a standardized format [69].

From this perspective, it seems that the evolution from blueprints to 3D digital models and the facilities that 3D modelers offer have come at the cost

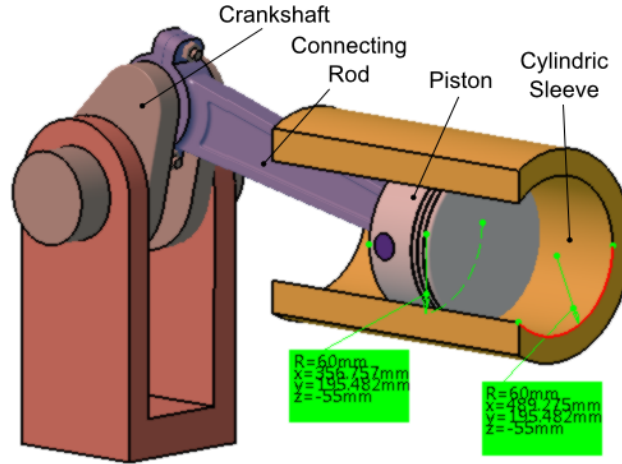


Figure 3.3: A 3D geometric model of a crank/piston mechanism. The piston and its cylindric sleeve are represented with the same nominal diameter of 60mm.

of the loss of reliability of any other information rather than approximate geometry. The bound between form, behavior and function that was seen in Section 2.3, is not available in modern DMUs.

Another observation to be outlined in this context is the lack of the thoroughness of these functional and technological annotations in 3D models, even when they do exist as Sections 1.6.2 and 2.6.2 showed. Many applications, in particular structural simulations, require the association of functional information down to the level of geometric interaction zones, e.g., contact surfaces between components. Those information are still not available in a DMU in a satisfactory manner that allows their involvement in the FEA preparation process, or any other application that would fit into a PDP and use assembly models.

Although DMU representations leave room for unconstrained textual annotations, that designer may utilize at different geometric levels, i.e., surfaces, solids, etc., to augment the model with functional and technological information, these annotations are too loose to provide any viable knowledge, as shown in Section 1.7. In fact, the best that we can expect from these annotations is to be coherent enterprise-wise. Reaching this cohesion however, requires engineers time and energy that compare to those needed for the manual pre-processing tasks outlined in Section 3.3.1. Method dependent on such a cohesion [32, 148] have therefore failed to provide a reliable connection to functional properties down to the level of component boundaries.

In the proposed approach, its purpose is to minimize the human inter-

vention during the FEA pre-processing stage, integrating domain knowledge in an inference system that enriches a pure geometric model with technological and functional information necessary for its adaption under the user-specified simulation objectives. This work contributes to a collaborative effort in the framework of the ROMMA project [1] to reduce the FEM preparation time to adapt CAD assembly models derived from DMUs into FE models.

In order to enable the semantic enrichment of a DMU that is required by state of the art FEA preparation approaches, the broken function-behavior-shape link should be mended. This recovery happens at three levels, as follows.

The functional interface level

Function is a result of interactions between components at their interface level. FEA applications need to know what functions are fulfilled by a component with regard to its neighboring components, in order to represent these functional interactions geometrically in a simplified manner, and decide what hypotheses can be made in the light of simulation objectives.

A mature approach to functionally supplement the DMU for FEA applications should thus consider the labeling of functional interfaces between components in an assembly. This leverage also requires the isolation of these interfaces as geometrically independent entities, to allow clearly interpretable labeling. Such labeling will be preformed on the basis of reference configurations between components referred to as ‘conventional interfaces’. *Conventional interfaces* and *functional interfaces* are introduced in more details in Chapter 4 to produce a taxonomy of functional interfaces as an explicit basis from which reasoning mechanisms will take place. To efficiently support the FEA preparation process, these interfaces need to be located accurately over the boundary of DMU components. This objective is addressed in Chapter 5.

The functional unit level

Each component in an assembly plays one major well-defined functional role within its functional group or groups. Before enabling geometric simplifications, this role should be outlined as it orients the content of suggested transformations.

To this end, a fruitful method must classify components into functional classes that deterministically define their functional role. Such classes, referred to as *functional designations* are introduced in more details in Section 4.2.3 and are based on particular spatial setup of *functional interfaces*. Indeed, *functional designations* are the major

result of the proposed approach. Enriching a component with *functional designations* from *functional interfaces* needs a reference to the behavior of this component (see Section 3.3.2). Indeed, this objective is addressed using a qualitative reasoning process as described in Chapter 6 that is followed by a rule-based reasoning (see Chapter 7) to infer the *functional designation* of this component. The purpose of these qualitative analyses and rule-based reasoning is to resolve the multiple interpretations that derive from the DMU input as pure geometric model. Through this approach, the objective is to set up a more generic approach than KBE ones (see Section 2.6.3) that takes advantage of the functional interface level to tightly link 3D geometry information to functional one.

The functional group level

In an assembly, a function is satisfied through physical interactions between a set of its components. Consequently, we can refer to these functions as *internal functions*. On a complementary basis, this assembly is characterized by functions with respect to its environment, i.e., these functions are often referred to as primary, secondary and constraint functions. Here, the functions referring to the environment of the assembly fall out of the scope of the present approach.

Efficient methods of geometric preparation for simulation purposes use such groups of components as *patterns* that indicate an entry point to relate geometry to functionality. As an example, Section 3.3.1 has referred to bolted junctions that designate a group of components that contain a screw, a nut and some tightened components, at least (see Figure 3.2). Processing such subsets of an assembly in an efficient manner connects with functional information when a selection process matters. Indeed, the first step to prepare a bolted junction for FEA is the selection of the corresponding components.

Accordingly, a beneficial enrichment of a DMU can organize components into functional groups that perform a given function. Those groups can then be labeled by the type of function they deliver. Labeled functional groups, referred to as *functional clusters* are an outcome of the proposed approach and can efficiently contribute to the desired component selection process as needed for FEA preparation. This objective is addressed in Chapter 8 and illustrated through a template-based selection process.

Figure 3.4 shows how functional annotations apply at the three aforementioned levels on the DMU of a centrifugal pump.

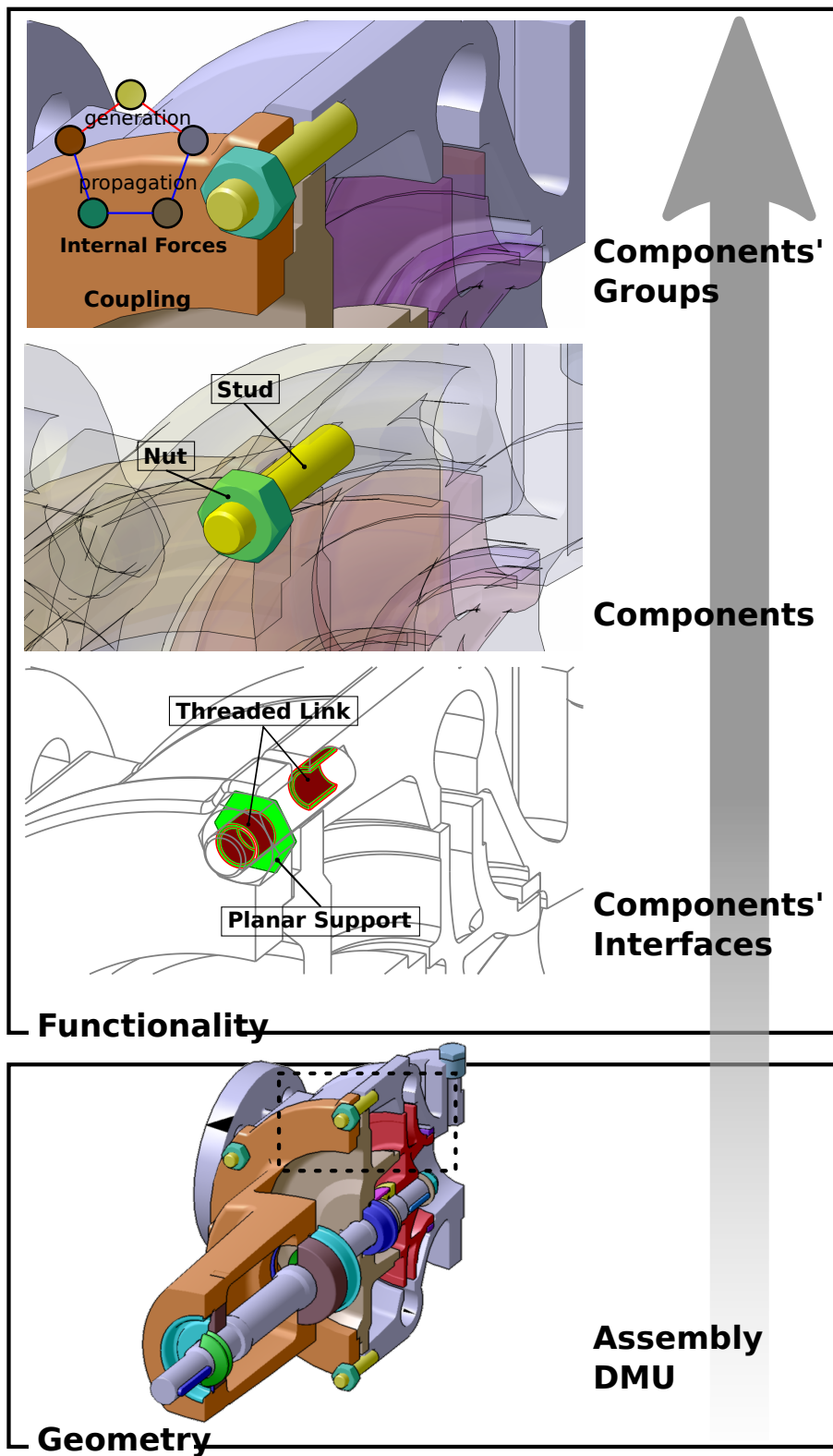


Figure 3.4: A synthetic, bottom-up approach to collect functional informations of a product at different levels, based on its geometric representation provided by its DMU.

3.5 Conclusion

In this chapter we showed that, in spite of its potentials, the DMU content as it is represented in today's industrial examples is not yet ready to enable its active participation to the preparation process of FEA. On one side, this is because of the *shape differences* between the real product and its digital representation, shown in Section 3.2. On the other side, another obstacle to the DMU utilization in simulation purposes is the *semantic gap*, shown in Sections 3.3 and 3.4, that prevents 3D component geometry from being connected to functional and technological annotations required for any robust and efficient geometric transformation.

Section 3.4 showed that those gaps should be bridged at three levels, namely the functional group, the functional unit, and the functional interface levels to allow the DMU to play its polymorphic role in the PDP as discussed in Section 1.10. The same section also showed that this task is currently being done manually, in a tedious and time-consuming manner. Therefore, this has introduced the major objective of the proposed approach toward the automation of tasks during DMU preparation for FEA.

Section 2.7 showed that current efforts in the field are still unable to feed the functional enrichment required by geometric transformation methods while taking into account today's industrial practices and conventions. Providing a method to automatically fill this need is one of the more precise objectives set throughout this chapter.

While this chapter has outlined the problems and set the objectives of the proposed approach, the following one presents starts presenting the proposed contribution and conceptualize our approach.

Chapter 4

Functional Restructuring and Annotation of Components Geometry

The proposed approach builds upon the relationship between function, behavior and shape shown in Section 2.3 in order to extract functional information from pure geometry of components for FEM preparation purposes as shown in Section 3.4. Reference states and design rules are introduced to express the behavior of components through a qualitative reasoning process and to complement the assembly geometric model used as input (see Section 3.3.2). These facts and rules reflect the domain knowledge, and enable to check the validity of certain hypotheses that must hold true at a specific state of the product, such as operational, stand-by or relaxed states.

Shortly, this bottom-up process starts with the generation of a graph of interfaces between components. Interfaces are initially defined geometrically. They are then populated with physical behavioral properties suggested by the geometry, producing a number of possible interpretations. The validation against reference states, reduces this number to ideally one interpretation per interface.

Once components interfaces are identified functionally, domain knowledge rules are applied to group the semantics of those interfaces into one functional denomination per component, and to cluster components into functional groups.

As a first step inside this overall process, the purpose of this chapter is to define some initial concepts related to component interfaces, their functional designation and the corresponding taxonomy. From these concepts, the above outline of the bottom-up approach to the functional enrichment of DMUs will be detailed

into an overall schematic description as a guide to the major steps that will be detailed in Chapters 5, 6, and 7.

4.1 Qualitative bottom-up approach

Section 2.3 has shown that the link between shape, behavior and function has been well established in the literature [73, 23, 173]. Design methodologies have been built upon this link to boost assembly design and collaborative product development [147, 142], while in another application of this relationship, top-down approaches are suggested to augment DMUs with functional attributes [146, 103]. These approaches, however, failed to functionally interpret commonplace geometric conventions with respect to FEA needs (see Section 2.4.2).

The present work proposes a bottom-up approach that takes the pure geometric representation of an assembly as an input. Elementary facts about geometric interactions are first collected. Those facts are then used to induce higher level knowledge about components and components groups in a synthetic manner, as Figure 3.4 shows.

Our approach is purely qualitative in a sense that no numerical values are used across the analysis process apart from the geometric parameters of the components, which are the input data. Geometric quantities such as distances are compared to each others, while no assumption about referential values or thresholds are made. This also applies to physical quantities which are described only symbolically with no precise values. This makes our reasoning universal, and independent of the availability of such quantities. These characteristics clearly distinguish the proposed approach compared to KBE (see Section 2.6.3) where quantitative parameter bounds are part of the KBE to perform dimensioning processes.

This inductive method allows for the inference of technological knowledge about the product at different levels, starting from components functional interactions, up to their functional groups. As a prerequisite to simulation preparation tasks, the proposed process is performed after the design activity, independently of design choices, and as an automated procedure.

In the rest of this chapter, Section 4.2 defines the terminology that is used in the proposed approach. The goal is to bring precise conceptual frames that are applied to notions encountered across the rest of this document. Next, Section 4.3 gives a synthetic description of our method, preparing the ground for in-depth development in chapters to come.

4.2 Common concepts

Throughout this manuscript we describe the proposed method using a terminology referring to concepts that are central to this research. Here, we

identify reference concepts contributing to all stages of the functional enrichment process, and define each as it applies to this approach.

4.2.1 Function as the semantics of design

Functionality is an example of non-geometric knowledge that a DMU still lacks, i.e., functions are essentially stated with natural language expressions. In fact, this knowledge becomes paramount when considering the preparation of a DMU for simulation purposes, as seen in Section 3.3. Section 2.2 examined different perspectives from which a *function* is seen in the literature.

In the context of our work, we join scholars that make clear distinction between function and behavior, while admittedly demonstrating the strong relationship that ties them [136, 73]. This perspective stands at a cross-road between teleological and behavioral viewpoints, as discussed in Section 2.2. Indeed, distinction between behavior and function is an important point in our approach since qualitative models of behaviors are set up and attached to components to infer component function. Consequently, this process relies on an effective distinction between behavior and function—otherwise the inference mechanism set up would be pointless—and a tight connection between them so that the qualitative behaviors can be effectively related to functions with meaningful inferences.

A function applies at different levels of the product structure. An interaction between two components delivers precise functionality that adds up to each component functional contribution. A particular function may be attributed to a subset of components that forms a group. As an example, the hydraulic pump illustrated on Figure 3.4 can be assigned a function to its whole set of components: (1) *move of a volume of fluid from the pump inlet to the pump outlet*¹. This function is also designated as the primary function of the product. Considering the group of components featured in Figure 1.5 that contains the hydraulic casing (orange), the two ball bearings (dark brown), the two elastic rings (black), this group can be assigned a function: (2) *guide the rotational movement of the shaft (gray)*. Now, considering a standalone component (see Figure 3.4 top), the stud (yellow) has as function: (3) *assemble together* the hydraulic casing (orange), the pump housing (gray), the hydraulic flange (brown), and the nut (green).

The function of the product is then satisfied as a result of functional groups collaboration. A component can be assigned more than one function and can contribute to several functions through different groups of components, e.g., the hydraulic casing (orange) is part of two groups of components defining functions (1) and (2).

¹The hydraulic pump is of type centrifugal for incompressible fluid. Therefore, its function reduces to displacement of a volume of fluid

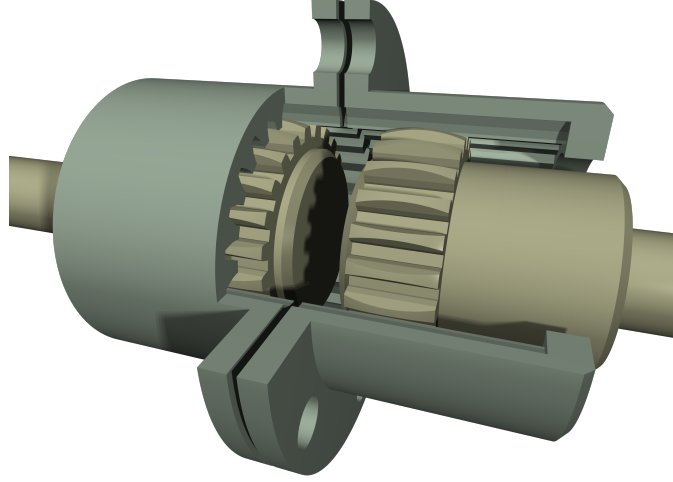


Figure 4.1: An example of a spline shaft-housing configuration that satisfies two functionalities that are axial positioning and power transmission. A cut in the housing component is made to show the coupling.

Definition 4.1 (Function). A function is a desired effect that a certain intentional configuration produces in a determinant manner.

In this sense, the underlying configuration is sufficient to produce the effect, hence, fulfill the function. However, it may or may not be unique, as some functions can be satisfied by several different means. For example, a *screw-and-nut* configuration satisfies the function of *tightening a set of components*. However, the same function can also be satisfied by other means, such as riveting, or even welding [104]. A screw-and-nut configuration is thus not necessary to tighten components, although it is sufficient.

In the same context, one configuration may fulfill more than one function, as it may produce more than one desired effect. An example is a *spline shaft-housing* configuration that satisfies both *axial positioning* and *power transmission* as functions (see Figure 4.1).

4.2.2 Functional Interface

Today's products tend to be *modular* [30]. Modularity has been established as an important paradigm in almost all design disciplines. Modular systems can be easily analyzed, tested, repaired² and upgraded. They also offer higher customizability as modules can be replaced to adapt to more specific requirements.

²In the context of mechanical products, repairing refers to the interchangeability of components.

An important aspect of modularity is *loose coupling* [128]. This means that each module of the product has only minimal knowledge about the other modules. System parts know about each others as much as necessary to get the system operational. In order to reach a loose coupling, a single module provides a minimal interface that interacts (couples) with other modules to fulfill the product global functionality, while the actual implementation of each module functionality is kept internal.

The same concept of interfacing appears in different engineering disciplines. In software engineering modularity consists in logically partitioning the software into different units at different levels, such as software packages and classes. Those units provide public interfaces describing what kind of stimuli they respond to. Communication between units is achieved through stimuli exchange, while internal implementation of each unit is kept private. In electronics, coupling coefficient refers to the amount of energy transferred between integrated circuits, and it is recommended to be the lowest possible in modular designs.

In mechanical engineering, modularity applies at different levels as well. Its first clear manifestation occurs at the component level, where a single component is meant to satisfy a very precise functional description. Another sign of modularity appears at the functional group level, where components are grouped to fulfill a higher level functional requirement, even though only partially with respect to the whole product functionality.

To allow the decomposition of high level functions into simpler ones, as suggested by modularity, mechanical components interact with each other through interfaces as well. Interactions can either be internal to a product or in connection with its environment. In this work we address the first category of interactions only, and we refer to interfaces that allow this interaction as functional interfaces (FIs).

A very basic example of a functionality that is fulfilled by FIs —and is usually kept implicit due to its triviality— is the relative positioning of components with respect to each others. Component shapes are designed so that they offer interfaces standing as obstacles to remove some of the degrees of freedom of some of their neighboring components.

In a real product, FIs are satisfied by functional contacts and plays (see Figure 4.2 for an example). Functions are defined by the geometric nature of the interaction between two components, and by their physical properties [118] (see examples in Section 4.2.5).

Definition 4.2 (Functional Interface). A functional interface (FI) is an interaction between two neighboring mechanical components that fulfills, or contributes to the fulfillment of a function.

An FI is characterized by its ability to propagate internal forces, with respect to the product as a physical system, between components involved

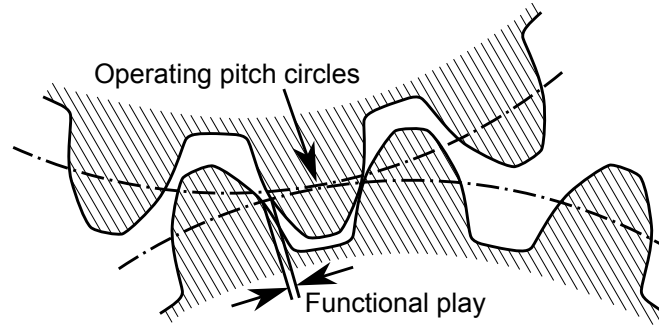


Figure 4.2: A gear train connection as functional interface, showing the functional play between gear teeth.

in the interface. A property that enables the FIs to meet its expected functionality from a dynamic standpoint.

When functionality is viewed from a kinematic standpoint, FIs are characterized by their abilities to restrict relative motion of components involved in the interface with respect to each others, reducing their respective DoF.

Examples of FIs are: threaded link, spline link, planar support, adherent conic support, etc.

4.2.3 Functional Designation

Besides loose coupling at the product level, modularity requires *tight cohesion* at the module level. That is, individual modules should fulfill one or more well-defined function each. Loose coupling and tight cohesion happen in parallel in a highly modular system.

When it applies to mechanical engineering, mechanical components are considered as modules, and tight cohesion reduces to assigning a number of precise functions to each component. There is a finite set of functions that a mechanical component can fulfill, considering an upper bound on the number of functions per individual component, the number of combinations is bound as well. However, not all combinations are common, and some, such as *tightening* and *guidance*, are not even possible.

We refer to the comprehensive set of functions that one mechanical component may satisfy as the functional designation (FD) of this component.

Definition 4.3 (Functional Designation). A functional designation (FD) is an equivalence class defined by the binary relation ‘*has the same set of functions as*’ that is defined on the set of all mechanical components.

Given \mathbb{C} the set of all mechanical components, and $\mathbb{F}^{\mathbb{C}} = \{f_1, f_2, \dots, f_n\}$ the set of all functions that any given component may satisfy. Let $c_1 \in \mathbb{C}$

and $c_2 \in \mathbb{C}$ be components and $\mathbb{F}_1^{\mathbb{C}} \in \mathbb{F}^{\mathbb{C}}$ and $\mathbb{F}_2^{\mathbb{C}} \in \mathbb{F}^{\mathbb{C}}$ the sets of functions that they satisfy, in respective order. We state that c_1 *has the same set of functions as* c_2 if and only if $\mathbb{F}_1^{\mathbb{C}}$ is equal to $\mathbb{F}_2^{\mathbb{C}}$:

$$c_1 \equiv_f^{\mathbb{C}} c_2 \iff \mathbb{F}_1^{\mathbb{C}} = \mathbb{F}_2^{\mathbb{C}}. \quad (4.1)$$

We note that $\equiv_f^{\mathbb{C}}$ is indeed an equivalence relation, as it is reflexive, symmetric, and transitive. We call $\equiv_f^{\mathbb{C}}$ the *functional equivalence* relation on \mathbb{C} ; the set of all mechanical components.

We note that if $\mathbb{F}_1^{\mathbb{C}} \neq \mathbb{F}_2^{\mathbb{C}}$, it logically follows from Equation 4.1 that $c_1 \not\equiv_f^{\mathbb{C}} c_2$. We thus infer that $c_1 \neq c_2$, since $\equiv_f^{\mathbb{C}}$ is reflexive.

Since an equivalence relation partitions a set into mutually exclusive equivalence classes [40], FDs, as equivalence classes of $\equiv_f^{\mathbb{C}}$ according to Definition 4.3, are indeed mutually exclusive sets.

This means that components having a functional designation $\mathbb{F}_1^{\mathbb{C}}$ are functionally different from components having a functional designation $\mathbb{F}_2^{\mathbb{C}}$. Indeed, $\mathbb{F}_c^{\mathbb{C}} \subset \mathbb{F}^{\mathbb{C}}$ is unique for a component $c \in \mathbb{C}$ and it is the identifier of its equivalence class. This identifier is expressed as a character string that uniquely characterizes $\mathbb{F}_c^{\mathbb{C}}$ and can be one of the following³:

- 1 An expression that relates to one function among the set of functions covered by $\mathbb{F}_c^{\mathbb{C}}$, e.g., a *stop screw*. Often, this expression relates to one major function of the component, its primary function;
- 2 An expression that uniquely designates $\mathbb{F}_c^{\mathbb{C}}$ in the common language, e.g., a *stud*, and implicitly matches $\mathbb{F}_c^{\mathbb{C}}$.

Figure 4.3 shows examples of selected mechanical components and their respective FDs.

FDs relate to FIs in a way that each set of functions at the component level (thus an FD) requires a set of functions at the interaction level (thus at least one FI), i.e. let $\mathbb{F}_c^{\mathbb{C}}$ be the FD of c , $\mathbb{F}_c^{\mathbb{I}}$ is the set of FIs belonging to c and $|\mathbb{F}_c^{\mathbb{I}}| \geq 1$. We refer to this one-to-many relation as the *functional breakdown*.

Unlike unreliable textual annotations (see Section 2.6), the concept of FD allows the qualitative reasoning and inference processes to give a component a functional identifier that unambiguously define the functionality of each labeled element. As shown in Section 4.2.6, FDs are organized into super-classes that contain each others, building a hierarchical functional classification. However, FDs are mutually exclusive classes at the leaf level, i.e., a given component can only belong to one FD. This labeling provides assembly components with a brief, yet precise functional description that can be, once assigned, exploited through out later stages of a functional analysis.

³The two categories highlighted may behave differently according to the language used, e.g. a ‘shoulder screw’ fall into category 1 in English whereas it is a ‘*axe épaulé*’ in French.



Figure 4.3: Functional designations exemplified by instance components of each (courtesy TraceParts [4]). The label under each instance(s) indicates the identifier used as functional designation for each equivalence class.

Geometrically speaking, this mapping restructures the geometry of a component that belongs to a given FD into interaction zones, each defining a FI, according to the functional breakdown of this component. Figure 4.4 shows an example of the restructuring of a cap-screw. This means that the functional breakdown is not a mere logical relation between a FD and FIs, there is also a strong connection with the geometric representation of the component c , i.e., the B-Rep of the 3D solid model of c must be decomposed into areas that match each FI of $\mathbb{F}_c^{\mathbb{I}}$. This connection efficiently sets up a consistency between the shape of c , its FIs, and its function with its FD. This consistency will be further enforced through the qualitative analysis and inference processes that will refer to the behavior of c , for the first one, and to the connection between shape, behavior and function, for the second one (see Chapters 6 and 7).

Also, it should be observed that the concept of FD is generic and not bounded to any quantitative parameter that may hinder its use for a component as it can happen for KBE approaches (see Section 2.6.3).

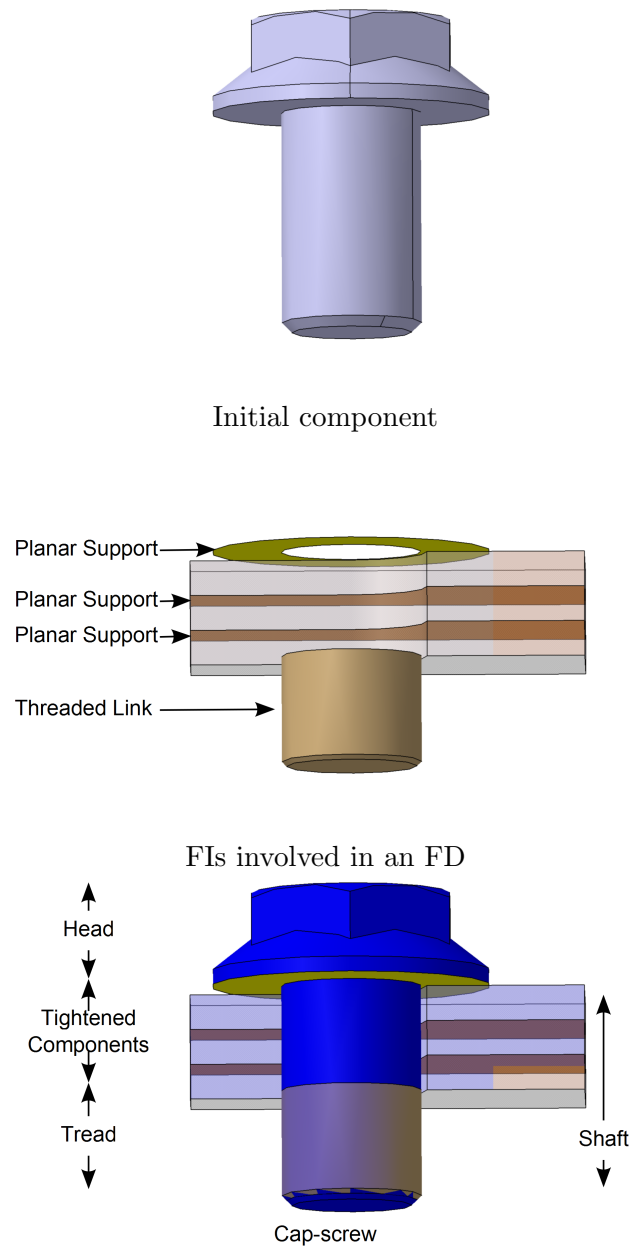
4.2.4 Functional Cluster

As mentioned earlier in Section 4.2.2; modularity can also apply at a higher level than individual components. A set of components may tie up together to deliver a coherent function or set of functions, while loosely interfacing with other components/groups of components through minimal interfaces. Those groups also form a module each. In the context of our research we refer to each such group of components as a *functional group*.

We refer to the set of functional groups that satisfies one or more particular functions as a functional cluster (FC).

Compared to the concept of FD where this concept can be stated for a standalone component c without referring precisely to geometric entities, it is critical to refer to the geometric interfaces between components when addressing FCs. Effectively, each function is not only characterized symbolically by its designation, e.g. *set screw*, but it is also instantiated through the geometric interfaces it involves between components. Therefore, from the set $\mathbb{F}^{\mathbb{C}}$ that symbolically represents all the functions any component can satisfy, we can derive $\mathbb{F}^c = \{f_i^c, f_j^c, \dots\}$ the set of functions c performs where \mathbb{F}^c is an instance of $\mathbb{F}^{\mathbb{C}}$ and f_i^c designates the symbolic representation of a function $f_i \in \mathbb{F}^{\mathbb{C}}$ associated with the geometric interfaces needed to describe f_i on c . f_i^c is an instance of f_i attached to c , an instance of a component class characterized by its geometric representation and its geometric interfaces with other components.

Definition 4.4 (Functional Cluster). A functional cluster (FC) is an equivalence class defined by the binary relation ‘*has the same set of functions as*’ that is defined on the set of all functional groups, \mathbb{G} . A particular instance



Structured component after matching FIs with an FD

Figure 4.4: A geometrically restructured cap-screw according to its functional breakdown.

of this equivalence class is g , a set of components. The set of functions of FC, \mathbb{F}^g , is achieved by more than one component and it is an instance of functions performed through some interfaces of components of g . At the difference of a FD that relates to a single component of a DMU, an FC addresses more than one component and not necessarily all the interfaces of each component of this FC, which means that a component of this FC can be involved into other FCs.

Let \mathbb{C}_d be the set of components contained in a DMU d , and $g \subseteq \mathbb{C}_d$ be a minimal non-empty set of components that together satisfy a set of functions \mathbb{F}^g . Each function of \mathbb{F}^g is associated with one or more interfaces between components of g . Based on that observation, we refer to g as a *functional group*. We observe that:

$$|g| > 1; \quad (4.2)$$

$$g \subseteq \mathbb{C}_d \subset \mathbb{C}. \quad (4.3)$$

Given \mathbb{G} the set of all functional groups and $\mathbb{F}^{\mathbb{G}} = \{f_1, f_2, \dots, f_n\}$ the set of functions that any given functional group may satisfy, $\mathbb{F}^{\mathbb{G}} \subset \mathbb{F}$, where \mathbb{F} designates the set of functions all assemblies can satisfy. Similarly to the observations mentioned previously about the set of functions associated with a component through its FD, \mathbb{F}^g is the instance of $\mathbb{F}^{\mathbb{G}}$ for the set g , which is an instance of a FC. Let c_i be any component of g and $\mathbb{F}_{c_i}^g$ the set of functions associated to c_i , i.e., $\mathbb{F}_{c_i}^g \subseteq \mathbb{F}^g$. It has to be observed that $\mathbb{F}_{c_i}^g$ does not necessarily contains all the functions attached to all the interfaces of c_i .

Now, let $g_1 \in \mathbb{G}$ and $g_2 \in \mathbb{G}$ be functional groups and \mathbb{F}^{g_1} and \mathbb{F}^{g_2} the sets of functions that they satisfy, in respective order. \mathbb{F}^{g_1} is associated with $\mathbb{F}^{\mathbb{G}_1} \subseteq \mathbb{F}^{\mathbb{G}}$, its symbolic counterpart. Likewise, \mathbb{F}^{g_2} is associated with $\mathbb{F}^{\mathbb{G}_2}$. We state that g_1 'has the same set of functions as' g_2 if and only if $\mathbb{F}_1^{\mathbb{G}}$ equals $\mathbb{F}_2^{\mathbb{G}}$:

$$g_1 \equiv_f^{\mathbb{G}} g_2 \iff \mathbb{F}^{g_1} = \mathbb{F}^{g_2} \quad (4.4)$$

We note that $\equiv_f^{\mathbb{G}}$ is indeed an equivalence relation, as it is reflexive, symmetric, and transitive. We call $\equiv_f^{\mathbb{G}}$ the *functional equivalence* relation on \mathbb{G} ; the set of all functional groups.

It is worth noticing that while FCs are equivalence classes, thus mutually exclusive, functional groups are not. Functional groups being reduced to a set of components c_i, c_j, \dots , the functions performed by these components would contain functions attached to several FCs because the set of functions attached to either of its components may contain functions related to more than one FC. In fact, functional groups of a given DMU are not equivalence classes because a component c can belong to two different functional groups,

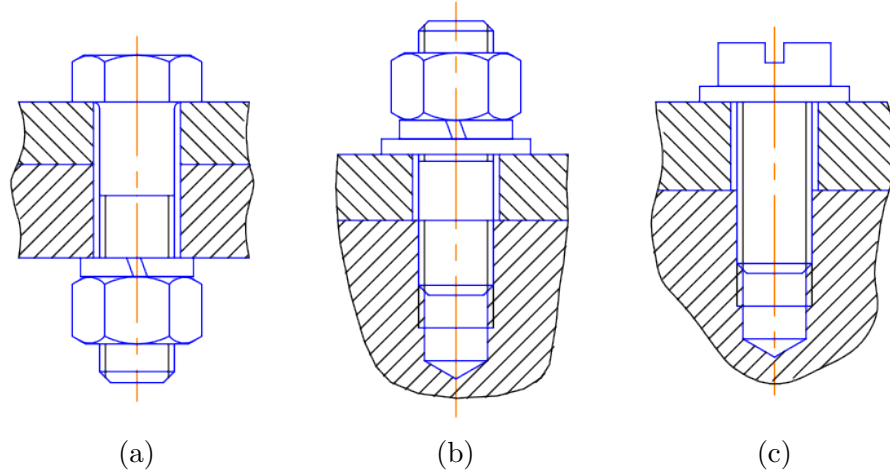


Figure 4.5: Examples of three functional groups belonging to the same FC: *disassemblable joint obtained with obstacles and threaded links*: (a) Bolted joint, (b) Stud joint (c) Screw joint.

g_1 and g_2 . Indeed, if $\mathbb{F}_{c_i}^g \subset \mathbb{F}_{c_i}$, it means that $(\mathbb{F}_{c_i} - \mathbb{F}_{c_i}^g) \neq \emptyset$ and this non-empty set of functions attached to c_i can be part of some other FC. This justifies an FC being identified by a set of functions rather than a set of components.

The identifier of a FC is expressed as a character string that uniquely characterizes $f_g = f'(g)$ and is usually an expression that uniquely designates f_g in the common language and may relate one function of f_g , e.g., a disassemblable joint obtained with obstacles and threaded links, and implicitly matches f_g . Here also, this identifier can be related to the primary function of the cluster.

To illustrate more precisely this concept, Figure 1.5 shows a set of components: hydraulic casing (orange), the two ball bearings (dark brown), the two elastic rings (dark blue), this group is assigned a FC whose identifier can be stated as: *guide the rotational movement of a shaft*, which refers to its primary function. Other illustrations of FCs are found in Figure 4.5 where each group refers to the same category of cluster that can be stated as *disassemblable joint obtained with obstacles and threaded links*. Indeed, each cluster is a variant of a technological solution that can be used to tighten components together using different categories of connectors, namely a bolt (Figure 4.5a), a stud (Figure 4.5b), or a screw (Figure 4.5c). In this example variants originate from the different connectors that belong to different FDs, respectively (capscrew and nut, stud and nut, capscrew) where each variant contains a different set of FIs.

Likewise FDs, FCs also relate closely to FIs. For instance, a functional

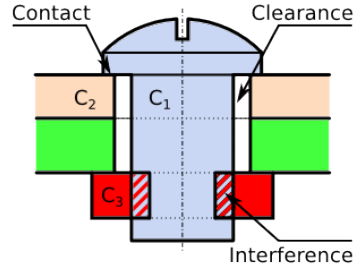


Figure 4.6: Examples of CIs as in a bolted joint.

group forming a *bolted joint* (an FC) can be recognized as the set of components that are involved in an internal load propagation cycle generated by a *threaded link* and propagated through FIs of types *planar* or *conical supports*.

FDs in their turn relate to FCs in what we refer to as *functional aggregation* where the union of functionalities offered by FDs produces a more general functionality characterized by the FC.

4.2.5 Conventional Interface

The functionality of an interface is decidedly determined by the geometric configuration of the interaction, and the physical properties of components materials, as briefly mentioned in Section 4.2.2. For instance a threaded part of a screw fulfills its function of *tightening* thanks to the helical shape of its groove and the friction along the thread that produces irreversibility. The inner tube of a tire does its job properly as a result of its toroidal shape and relatively low material stiffness.

Subsequently, geometric interactions between adjacent components reveal essential information that guides the identification of functional properties. Objects interactions in the digital model, however, do not accurately reflect reality, as previously demonstrated in Section 3.2. In fact, the way interactions are represented is no more than a convention made by designers, or prescribed by a company, or a common practice since there is no standard referring to the 3D representation of components.

We refer to interactions between neighboring components in a DMU as conventional interfaces (CIs). Physical, and functional properties can be attached to this concept. However, a CI is initially identified by a geometric interaction between two components in a DMU. This interaction encapsulates an *interaction zone* which can be either a *contact*, an *interference*, or a *clearance* (see Figure 4.6).

Contact A contact between two components C_1 and C_2 defines one or more

shared surfaces or shared curves, without any shared volume (see Figure 4.6). The interaction zone of a contact is defined by this set of shared surfaces and/or curves, leading to potential non-manifold configurations, i.e. a contact along a surface area connected to a contact along a line⁴.

A contact representation is usually realistic in the sense that a contact in a digital model may reflect the same configuration in the corresponding real product, where C_1 and C_2 are, indeed, touching each other. However, when a clearance between C_1 and C_2 becomes small enough in reality, it may conventionally be reduced to a geometric contact as well. Consequently, a cylindrical contact can be functionally interpreted either as a *loose* or a *tight fit* (see Figure 3.3 for an example). In some conventions a contact may represent an idealization of more complex settings, like threaded links or gears and sprocket connections.

Contacts provide very valuable information to our reasoning, as they usually help defining locations where resulting interaction forces can be transmitted. At the same time they work as motion barriers reducing components DoF.

Interference An interference defines a shared volume between two components C_1 and C_3 . Obviously, an interference is a non-realistic representation in the sense that the two digital shapes of C_1 and C_3 interfering in a DMU do not represent overlapping volumes of C_1 and C_3 in a real product, as this leads to non-physical configurations. Therefore, interferences are often the result of local shape simplifications combined with rather complex settings of component locations. For instance, engaged spur gears frequently result in cylindrical interference volumes (see Figure 3.1).

Also, when interferences become small enough, e.g., a shaft diameter that is slightly greater than its housing diameter to produce a tight fit between them, it is not represented in the DMU where the two corresponding components have the same diameter and produce a contact.

Due to their idealized nature, interferences are harder to interpret than contacts; however, they also provide valuable information about functional attributes of a CI.

Clearance A clearance occurs when a distance between surfaces of two components C_1 and C_2 is less than a defined threshold while staying greater than zero, i.e. C_1 does not touch C_2 in the area of the corre-

⁴Though this configuration is not mechanically meaningful, it can be a geometric configuration appearing in a DMU.

sponding surfaces. The distance value acting as a threshold between the two components is a matter of design decision.

The interaction zone of a clearance is the set of surfaces of C_1 and C_2 for which the minimal distance is less than the threshold while staying strictly positive.

When this minimal distance conveys a functional intention, the clearance is said to be a functional play (see Figure 4.2). As a convention, when a functional play becomes small enough, it can be represented as a contact in a DMU.

Definition 4.5 (Conventional Interface). A conventional interface (CI) is a conceptual entity that represents an interaction between two components in an assembly. It is identified by the geometric interaction of corresponding components in a DMU, and augmented with other semantics such as physical and functional properties.

Given \mathbb{I} the set of all CIs in a DMU. We define the binary relation ‘forms’ as $(\forall c_1 \in \mathbb{C}, i \in \mathbb{I}) c_1 \mathcal{R}_f i$ if and only if the component c_1 forms the conventional interface i with another component $c_2 \in \mathbb{C}$. As stated in the definition of a CI, this interface is defined from two components. If \mathcal{R}_f relates one component c_1 to a CI, i , this means that another instance of \mathcal{R}_f relates c_2 to the same CI, i .

We also define the binary relation ‘links’ as $\mathcal{R}_l = \mathcal{R}_f^{-1}$.

We note that each CI ‘links’ exactly two components. This can be noted:

$$\begin{aligned} (\forall (i, c) \in \mathbb{I} \times \mathbb{C}; \exists (x, y) \in \mathbb{C}^2) \quad & x \neq y \\ & \wedge x \mathcal{R}_f i \wedge y \mathcal{R}_f i \\ & \wedge c \mathcal{R}_f i \implies (c = x \vee c = y). \end{aligned}$$

Since FIs are the result of interactions between components, a CI can be seen as a potential FI, when the interaction it incorporates conveys a functional meaning. There exists no direct one-to-one mapping between CI and FI though. For example a *cylindric interference* can equally represent a *threaded link* as well as a *spline link*. This is due to the simplified nature of CIs. In both cases, either helical threads or meshed teeth and grooves configurations are represented as a simple interference. These ambiguities will be processed with the qualitative behavior to filter out some of them (see Chapter 6).

Section 5.4.1 shows how to interpret CIs into their functional counterparts, i.e., their corresponding FIs.

4.2.6 Taxonomies

The concepts previously defined in this chapter define equivalence binary relations, that is they divide the global sets of functions, components in-

teractions, components, functional groups, and geometric interactions into mutually exclusive subsets called classes. Examples are:

- Function defines *reversible tightening* as a class of the global set of all functions \mathcal{F} .
- FI defines *threaded link* as a class of the global set of all components interactions \mathcal{I}_f .
- FD defines *cap-screw* as a class of the global set of all components \mathcal{C} .
- FC defines *bolted joint* as a class of the global set of all functional groups \mathcal{G} .
- CI defines *complete cylindric interference* as a class of the global set of all geometric interactions \mathcal{I}_g .

Those classes, however, can be grouped in their turn into more general ones, i.e., larger mutually exclusive subsets. This grouping is the result of sharing some less discriminant semantic properties, e.g., functional or geometrical ones, across multiple primitive equivalent classes. We note that, for example, *reversible tightening* is no more than a *tightening* function which has a more specific property of being reversible. Thus, *reversible tightening* and *irreversible tightening* can be grouped in a more general function class called *tightening*. As a complement, *disassemblable joint obtained with obstacles and threaded links* is a more specific class that is indirectly related to the *reversible tightening* class. In the same spirit, *complete cylindric interference* and *partial cylindric interference* are grouped in a more general geometric interaction class called *cylindric interference*.

This leads to the structuring of each concept in a hierarchical structure that reflects this generalization relation. We call each of these hierarchies a taxonomy.

Definition 4.6 (Taxonomy). A taxonomy is a tree-like structure for which the root is the concept domain of discourse, and the leaves are equivalence classes that the concept defines. At each node, the children of the node are mutually exclusive sets.

A concept domain of discourse is the global set that the concept covers. That is \mathcal{F} for function, \mathcal{I}_f for FI, \mathcal{C} for FD, $2^{\mathcal{C}}$ for FC, and \mathcal{I}_g for CI.

Organizing FD in a hierarchical structure allows the proposed approach to gradually identify components. For instance, a given component can be first identified as a *fastener* at an early stage of the reasoning process, as it complies to certain rules, this can be refined further by identifying the component in hand as a *screw* in later stages. Finally, the component can be precisely assigned the FD of a *cap-screw* if certain conditions are met.

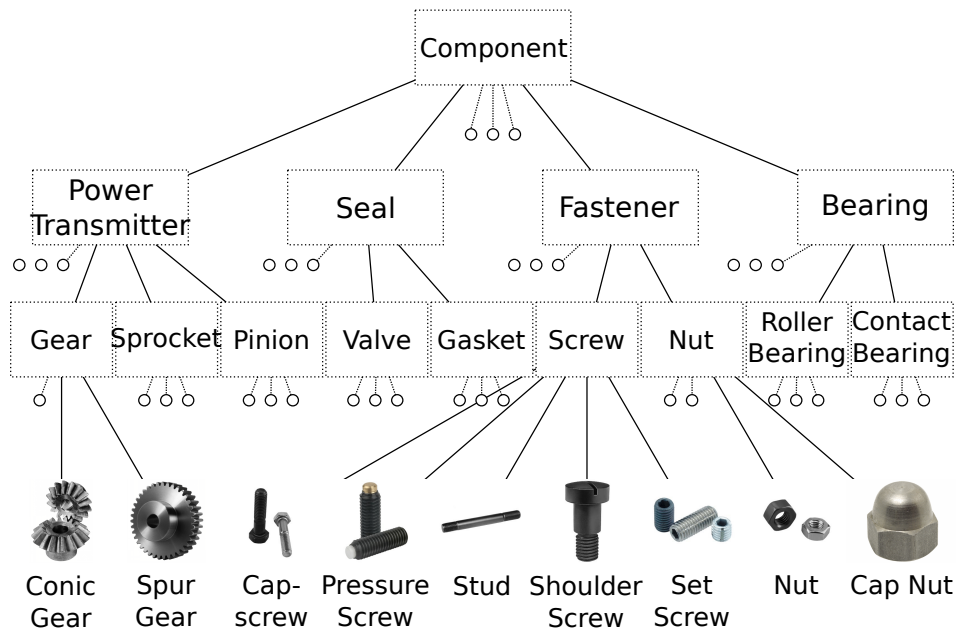


Figure 4.7: A subtree of the taxonomy of FDs.

Figure 4.7 shows a portion of the taxonomy of FDs, showing the path to *cap-screw*. It is now important to note that the taxonomy of FDs uniquely defines the components of a DMU where these FDs are located in the leaves of the taxonomy. It is effectively the place where the taxonomy associates a single component to one FD.

The same applies to CIs, where geometric properties of an interface can be narrowed down in an adaptive manner using the taxonomy of CIs, starting with detecting whether it is a *contact*, *interference*, or *clearance*⁵ down to the precise geometric configuration identification of the CI.

Figure 4.8 shows portions of FI taxonomy (left) and CI taxonomy (right), and the relation between each class expressed at the leaf levels. This relationship links each CI to all FIs that it may conventionally represent according to observations in industrial DMUs. As the figure depicts, and as shown in Section 4.2.5, this connection is inherently ambiguous as it relates one CI to possibly more than one FI. It however defines at the outset how CIs must be functionally interpreted knowing only their geometric properties (see Section 5.4). This ambiguity is reduced on a case-by-case basis, as shown in Chapter 6.

⁵We show later in Chapter 5 that we are only interested in contacts and interferences in the scope of our research.

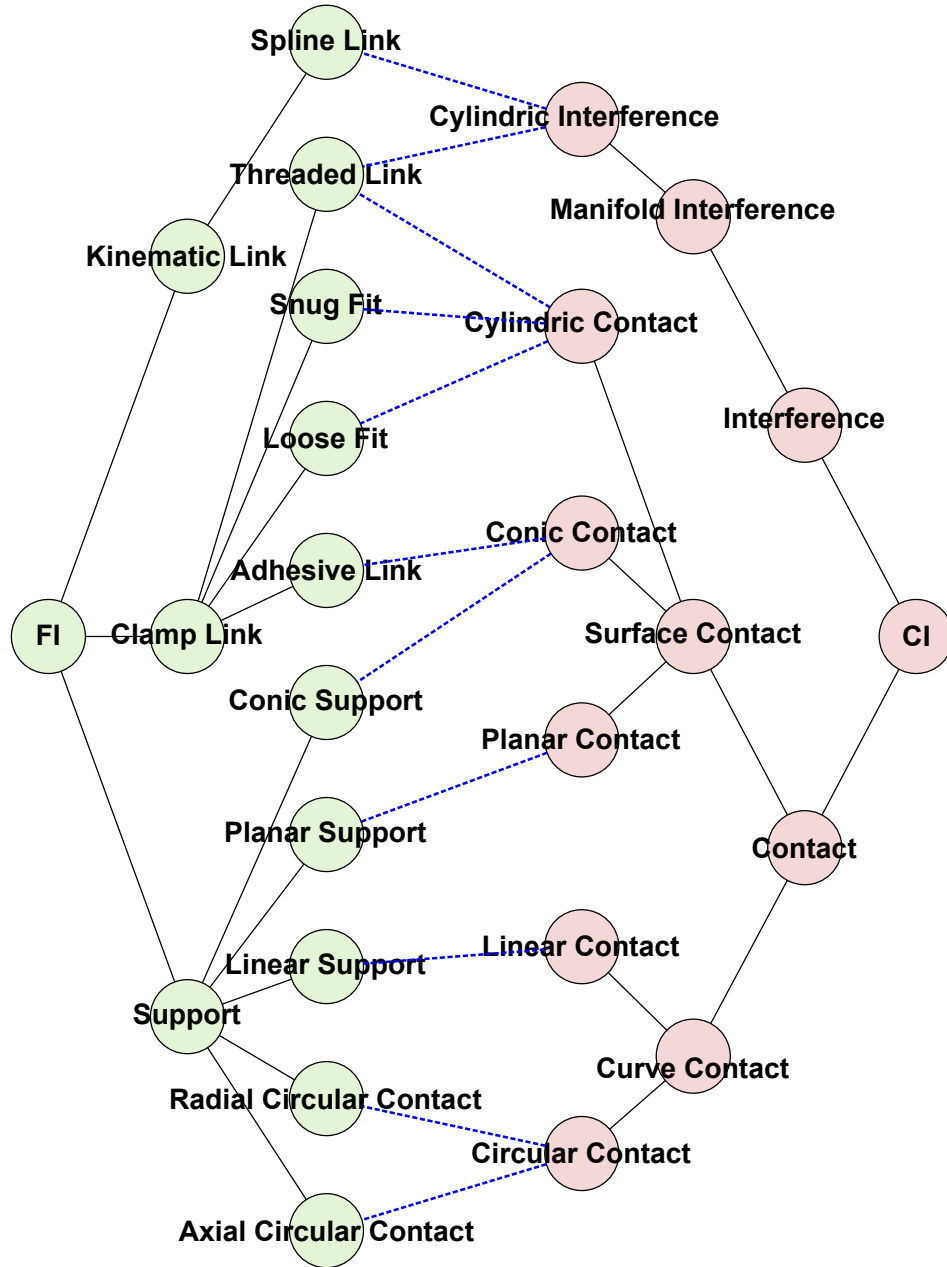


Figure 4.8: Taxonomies of FIs (left) and CIs (right). Dotted lines show how they relate to each other at the leaf level (the figure only shows a partial view of each taxonomy).

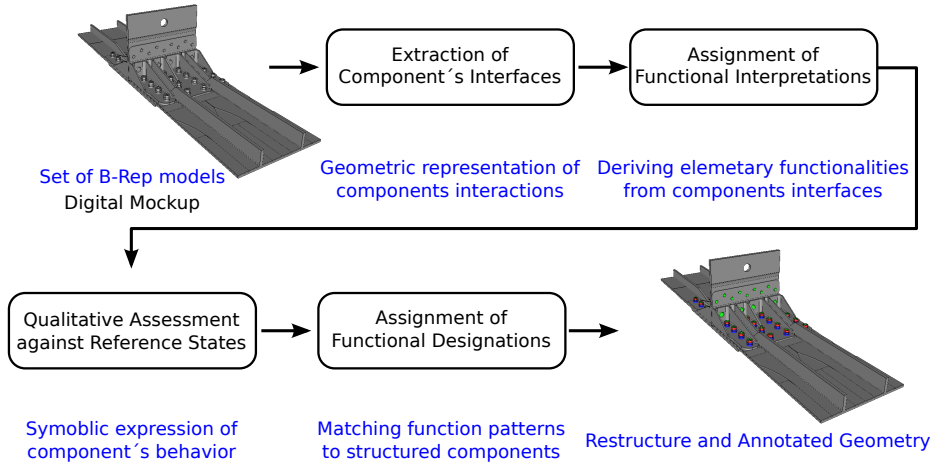


Figure 4.9: Data and process flow diagram of the proposed approach.

4.3 Method walk-through

In this section we define the outlines of our method using the reference concepts introduced in the previous sections. The method takes as only input the geometry of product components as represented in a DMU. The proposed approach treats this model through different stages, as shown in Figure 4.9, and concludes to deliver a restructuring of the initial geometric models of components, with coherent technical and functional annotations at different level of details where the first level is the FD of components.

Next, we briefly present each of these stages to synthesize the overall process before details are elaborated in a dedicated chapter per stage in the rest of this document. As depicted on Figure 4.9 the overall scheme is of type linear and its main stages are as follows.

Extraction of component interfaces

The first step of the process is purely geometric. In this phase geometric interactions between components are detected. A CI is created for each valid interaction, and it is identified by the geometric nature of the interaction zone. CIs are also loaded with information about their location and orientation with respect to the whole assembly. Based on the nature of each CI, the taxonomy of CIs (see Figure 4.8) is populated and a logical connection is set up between the geometric data structure of this CI and its associated instance in the taxonomy.

Also, the binary relations between components and CIs are expressed in a graph structure, which is passed to the next stage together with the

taxonomy of CIs, as a result of the geometric analysis.

Assignment of functional interpretations

In this stage, a first attempt to functionally interpret CIs is made. This is performed strictly using the intrinsic geometric properties of each interface. Generally speaking, more than one functional interpretation is possible per geometric configuration, hence, more than one possible FI is assigned to each CI. This process populates the taxonomy of FIs and set up the connections between this taxonomy and the taxonomy of CIs that has been populated at the previous step. The connection between the taxonomies conforms to Figure 4.8. At this stage, all the functional interpretations of the interfaces between components are expressed and structured to characterize the ambiguities living in a DMU that originate from the conventional representations applied to each of its components.

Generating the instances of the taxonomy of FIs sets a link between shape and function at the interface level between components. Because the interfaces are the most elementary areas where components interact, their functions are elementary and the corresponding set of functions is rather small and can be enumerated easily. The behavioral phenomenon used to assign function to each CI is based on the kinematic behavior of the interface, i.e., the relative movements between the components defining this interface.

Chapter 5 details the detection and initial interpretations of geometric interactions.

Qualitative behavioral assessment

In order to reduce the number of function interpretations to one per interface between components, a physical dimension is given to each CI.

Since physical properties of interfaces are not yet available, we generate assumptions for each possible interpretation, then the goal shifts to refute some of those assumptions, thus their relative function interpretation. These physical properties are related to a behavior of the DMU that express a transition between reference states assigned to the DMU (see Section 2.2 and Section 2.3).

This refutation is made by validating each possible interpretation and checking its physical plausibility and mechanical meaningfulness against a set of established reference states. The corresponding process is based on a qualitative behavior simulation to be independent of physical quantities that are not available with the DMU and/or not available at the stage of a product development process where the DMU is used as input.

The output of this stage is a precise functional interpretations of CIs in terms of their respective FIs.

Chapter 6 describe the details about the qualitative analysis algorithm as well as the concept of state.

Assignment of functional denominations

Once functional properties of each interface is identified, i.e., the previous step has discarded all unnecessary interpretations to keep only one of them per interface, the functionality of components is investigated based on this knowledge.

This is done using rules that describe relations between FDs, FCs, and FIs, in a pattern-matching-like manner. This is the stage where relationships between shape, behavior and function is used (see Section 2.3). The spatial layout of interfaces combined with their functional behavior obtained from the previous stage is used to infer the appropriate function of some components, hence their corresponding FD (see Section 4.2.3).

The result of this stage is components classification into their corresponding FDs, and components clustering into FCs. The component clustering derives from the taxonomy generated from FCs.

Chapter 7 provides a detailed description of this rule-based reasoning.

Semantically-augmented geometric model

The previously collected knowledge is finally integrated into a semantically-enriched and restructured geometric model of the DMU components.

The restructuring is the result of the breakdown of model components into geometric entities that reflect the functional interactions between components by means of its CIs (according to the functional breakdown), and the classification of components belonging to the same functional cluster into groups (according to the functional aggregation). Additionally, the geometric decomposition of components can be used to describe precisely the FDs of components, i.e., the FIs of components involved in the FD of a component are also connected to each other and related to the FD of this component.

The semantic enrichment is achieved by the functional annotation of interfaces, components, and groups of components. Based on this enrichment, advantages can be gained to select components in a DMU in accordance with their FD and their function and process their neighborhood (see Chapter 8). This is particularly relevant for the pre-processing of DMUs for FEA.

4.4 Conclusions

This chapter has introduced some reference concepts of the proposed approach. These concepts outline the knowledge modeling process used throughout the proposed approach. There, the dependencies between shape, behav-

ior and function is precisely analyzed to produce efficient mechanisms that can be used to enrich a purely geometric model of a DMU up to functional information.

It has to be observed that the proposed approach is not depending upon a particular morphology of the components that could restrict the range of DMUs that could be processed. Therefore, the proposed enrichment process is more generic than KBE approaches (see Section 2.6). Also, the boundary decomposition of components resulting from the identification of CIs and their enrichment with functional information to obtain FIs show how this process can contribute to the definition of functional features and how these features differ from features encountered in prior work (see Section 2.4). Here, the dependency between shape, behavior and function brings consistency to the functional information obtained from the enrichment process.

The proposed constructive bottom-up method has been presented synthetically to emphasize its key steps. It has outlined central concepts (see Section 4.2) and how they contribute to the context of the proposed enrichment process, before we enumerated the major stages of the proposed method in Section 4.3, shedding lights on how these concepts interact and fit into the paradigm of shape - behavior - function dependencies. Introduced concepts are revisited in the upcoming chapters, where we develop our approach in more details.

Chapter 5

Functional Geometric Interaction between Components in a DMU

Product modules interact through precise interfaces, this applies to all engineering domains as we have shown in Section 4.2.2. When it comes to mechanical engineering those interfaces are the direct result of components geometric interaction, where components play the role of modules in this context. This joins what is referred to in the literature as form-function relationship and shown in Section 2.3.

The first indicator to components functions thus is their geometric interactions. In this chapter we show how to efficiently, yet precisely detect those interactions of interest as basis of a thorough functional analysis.

5.1 Functional surfaces

As demonstrated in Section 4.2.2, FIs occur at the geometric interactions between components in a DMU such as contacts and interferences. This can theoretically happen between any kind of surfaces at both sides of the interface, resulting in different possible types of interaction zones.

Observation shows, however, that geometric interaction of interest to our analysis, that is those who convey a functional meaning, are restricted to a subset of all possible configurations.

In fact, studying industrial DMUs showed that functional interaction happens at parts of the components that fall in one of the following two categories.

- Simple geometric configuration in the real product such as planar con-

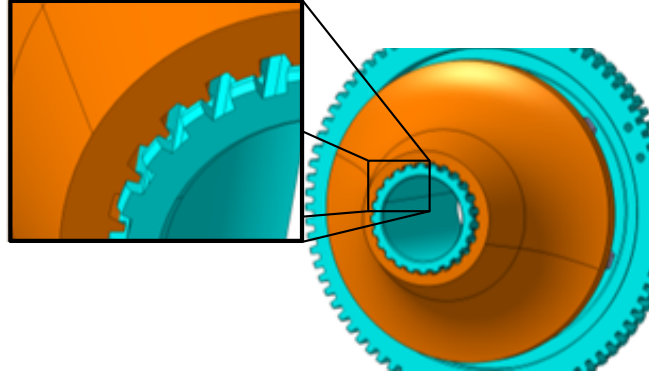


Figure 5.1: Approximate relative rotational position of components in a spline link; detailed view on the left, global one on the right (courtesy AN-TECIM).

tacts and cylindric fits. In this case the real geometry is not simplified in the digital model, and surfaces are represented as they should be manufactured.

This configuration is quite common. It is preferable as planes, spheres, and ruled surfaces are relatively easy to machine rather than free form surfaces.

- Complex repetitive geometric features, like helices and involute teeth. Such profiles are necessary to insure specific physical behavioral properties, allowing the fulfillment of particular functions.

A detailed representation of such surfaces potentially leads to inaccurate relative positioning of components when they are assembled together (refer to Figure 5.1 for an example). Moreover, the machining of such profiles are done independently of the digital model, since whole components are often out-sourced, or features are machined using particular tools. For these reasons, such detailed configurations are simplified in the DMU, and reduced to simple contacts or interferences, as shown in Section 3.2. Figure 1.7 shows how a threaded connection between a bolt and a nut is represented as a simple interference.

In the lights of the aforementioned observation, we note that only interactions that occur between canonic surfaces in the digital model may hold functional interpretations, hence, only these interactions are of interest to our research. This leads to the following hypothesis.

Hypothesis 5.1 (Functional surfaces). In a DMU, FIs are represented using canonical surfaces that can be either planes, spheres, cones, tori, or cylinders. We refer to such surfaces as *functional surfaces*.

Free form surfaces, such as Bézier patches [151] and NURBS [130], are still used in product modeling for different reasons, not the least of which are their precise mathematical representation, intuitive modeling, and their agronomic and aerodynamic qualities.

Since free form surfaces do exist in a DMU, interaction between them may indeed occur in an assembly. However, they usually do not connect components functionally. They may though represent function interactions with the product environment, such as aerodynamic drag. Nonetheless, such interaction are out of the scope of our study as external elements such as gases and fluids are usually not present in a DMU.

5.2 Geometric preparation and rapid detection of interactions

In this section we describe the very first stage of our method which consist of the geometric analysis to detect components interfaces and their geometric properties. First, the nature of the method input as the geometric model of the product is explained. Before going into details of the optimized detection algorithm.

5.2.1 Geometric model as global input

As mentioned in Section 4.3, our analysis and reasoning to reveal functional properties about the product are based solely on the geometric model of this product, as represented in its DMU.

In the context of our research we opt to use a standardized portable representation, that is widely used in industry to communicate product models between different CAD systems. This is STEP format as standardized by ISO 10303 [7].

STEP aims to provide a neutral format that represents product data all along its lifecycle, across different platforms. However, and as shown in Section 3.4 industrial practices make little use of STEP support of non-geometric annotations. We are, thus, only concerned about part of the standard that deals with geometric representation and referred to as AP 203 [8].

STEP is implemented using one of the following methods.

STEP-File Where product data are represented in and ASCII structured file. This method is defined by ISO 10303-21 [10].

STEP-XML An alternative to the previous method that uses an XML structured file. Both methods have the advantage of being highly portable across different platforms. This method is defined by ISO 10303-28 [9].

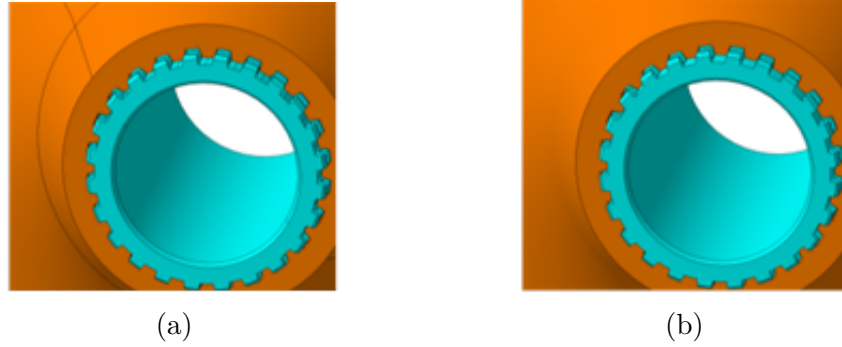


Figure 5.2: Effect of maximal faces and edges generation. Patch boundaries are marked with black edges. Initial boundary decomposition (a), boundary decomposition with maximal faces and edges (b) (courtesy ANTECIM).

SDAI An Application Programming Interface (API) that deals with products data, and is defined by ISO 10303-22 [11].

Our methods takes a product DMU represented as a STEP-File formatted file, and passes it to the first stage of our approach: the geometric analysis.

5.2.2 Maximal edges and surfaces

STEP describes components geometric models using a B-Rep format. Unfortunately, B-Rep encoding of a geometric object is not unique. That is; two STEP files may represent the same shape differently. This is due to the fact that an edge (then called a wire) can be represented as a set of topologically-connected smaller edges laying on the same curve. The same applies to faces, where a face can be divided into smaller ones that share the same surface and are topologically-connected. This phenomenon originates from component modeling process where functional surfaces are often broken down into smaller pieces because of the constructive nature of the process inherent to industrial CAD modelers.

Additionally, geometric modelers are subjected to topological and parametric constraints [111]. This prevent the boundary decomposition from matching the real boundaries of a component. For example, a cylindrical surface can be represented either with two half cylinders or a single cylindrical patch whose boundary contains a functionally meaningless generatrix, since it is not a boundary of any surface on the real component (refer to Figure 5.2).

The removal of such unnecessary geometric elements is mandatory to obtain a unique geometric representation of a shape. As shown in Sec-

tion 2.5.2, representation uniqueness is a must when useful information is to be extracted from a geometric model. As a desirable side effect, the decrease of the number of geometric elements boosts the performance of all subsequent geometric treatments.

To obtain this representation of components boundaries, adjacent faces (i.e. topologically-connected ones) that belong to the same analytical surface are merged into one entity; a *maximal face*. A maximal face is represented by its underlying oriented and topologically-connected faces. Edges are also grouped into *maximal edges* using the same criterion, where adjacent edges laying on the same analytical curve are merged. A maximal edge is represented by its underlying oriented and topologically-connected edges. As a result, a cylindrical face can end up with a boundary described by two closed edges without vertices. The corresponding data-structure uses hyper-graphs and was introduced by Foucault et al. [68].

The resulting normalized geometric model with maximal edges and surfaces has a minimal number of topological elements (vertices, edges, and faces).

5.2.3 Geometric interaction detection

Once the geometric model is normalized, it makes way for the detection of geometrical interaction zones of interest that define CIs. This means the detection of contacts and interferences that potentially convey a functional meaning.

Clearances as functional plays

Clearances may imply functional intention as shown in Section 4.2.5, this is, however, more intricate to detect than contacts and interferences. This is basically because clearance detection, unlike that of contacts and interferences, is dependent on a parameter which is the *play threshold*. The play ρ is the minimal distance that two component preserve between their surfaces, and can be defined as

$$\rho = \min_{p_1 \in \partial C_1} \min_{p_2 \in \partial C_2} \|\overrightarrow{p_1 p_2}\|$$

as shown in Figure 5.3. If the play between two components is less than or equal to a predefined threshold $\rho \leq P$, components C_1 and C_2 are said to have a clearance.

This dependence on an input parameter is contradictory to our assumption of a purely qualitative reasoning (motivations are explained in Section 4.1). Moreover, and in spite of the potential functional implication of plays, they are of less use to later stages of our analysis, as this potential functional contribution is not easy to verify. Finally, functional plays are usually not simplified in any way when preparing a CAD model for

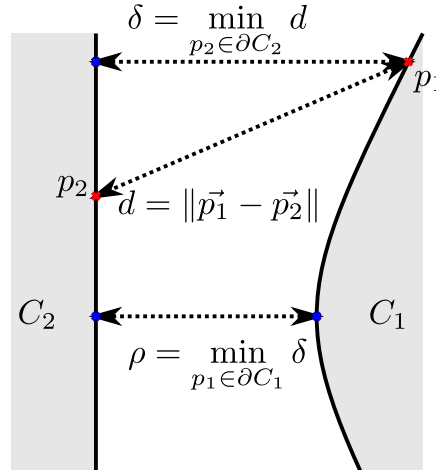


Figure 5.3: Calculation of the play ρ between two solids C_1 and C_2 , where d is the distance between two given points on the surfaces of C_1 and C_2 , and δ is the minimal distance between a given point on C_1 and the surface of C_2 .

simulation, that makes their identification irrelevant to the geometric transformation process.

For all these reasons, clearance detection is kept out of the scope of our implementation, while focus was given to efficient detection of contacts and interferences.

Early elimination of negatives

A naive approach to the geometric interaction detection is to use boolean operators—explained hereafter—between pairs of solids of the model at hand. This can however be enhanced using the early elimination of negatives.

Early elimination of negatives filters out candidate pairs that obviously have no interaction zones, this is done using *bounding boxes* technique. A bounding box of an object C is a minimal box with edges parallel to the global coordinate system, which inclusively contains the object C . Each bounding box is then defined by six values (in 3D): x_{min} , x_{max} , y_{min} , y_{max} , z_{min} , and z_{max} . Bounding boxes interaction check reduces to 6 floating-point comparisons at most. If the bounding boxes of two components do not interact geometrically, the two components do not either.

Boolean operators drawbacks

Boolean operators provide an accurate tool to detect contact and interference zones. The intersection between two solids¹ C_1 and C_2 is computed, generating a set of geometrically connected shapes \mathbb{S} , where

$$\begin{aligned} \forall S_1, S_2 \in \mathbb{S}, \quad \text{int}(S_1) \cap \text{int}(S_2) &= \emptyset \\ \bigwedge \quad \bigcup_{S \in \mathbb{S}} S &= C_1 \cap C_2 . \end{aligned}$$

where $\text{int}(S) = S - \partial S$ is the interior of the solid S .

If \mathbb{S} is an empty set, the two solids are said to have no geometric interaction. Otherwise, and for each resulting shape S , if the shape is a volume (possibly with non-manifold configurations) the two solids are said to be interfering at S . If the shape is of a lesser dimension (a surface, curve, or point) the two solids are said to be in contact at S .

Nevertheless, this tool is costly in time and resources. This cost quickly becomes prohibitive, even for modestly large DMUs. Additionally, even though the precise interaction zone is obtained as a result of the boolean operation, geometric parameters are still to be looked for into those resulting shapes. That means that the cylindric interference between two object, for instance, is returned as a B-Rep shape, and still needs to be studied to obtain the axis and diameters of the interaction, the axis being notably required for further stages of our approach.

Canonical face comparison

To avoid the burden of boolean operators when not needed, a simpler, yet more efficient, detection technique is utilized, based on the mere comparison of geometric entity of two neighboring objects. We consider two objects to be *neighbors* if they pass the bounding boxes test (i.e. their bounding boxes interact with one another).

In this sens, bounding boxes are used first to filter out non-adjacent solids. The remaining ones are then checked pairwise for geometric interactions.

For each pair of solids, maximal faces of one solid that lie on canonic surfaces are compared against those of the other. We adopt a simple, yet extensible approach to extract geometric interactions, based on the comparison of the geometric characteristics of canonic surfaces. This comparison is no more than a secondary filtering of irrelevant interaction candidates surviving the bounding boxes check. The final detection of interaction zone is discussed in Section 5.3.

Take two solids C_1 and C_2 whose bounding boxes are in interaction. Therefore, canonic faces of C_1 is to be compared to these of C_2 . Let us

¹Solids here are represented by their closure $C = \text{int}(C) \cup \partial C$.

consider a maximal faces $F_1 \subset \partial C_1$ and $F_2 \subset \partial C_2$. We need to compare F_1 to F_2 .

If both F_1 and F_2 are planar, we check whether normals of the two faces are opposite to each others, and if the difference between the position vectors of the two faces is orthogonal to the normals. Given \vec{n}_1 and \vec{n}_2 the normals of F_1 and F_2 , and \vec{p}_1 and \vec{p}_2 their respective position vectors, the above-mentioned check can be stated as follows;

$$\begin{aligned} \vec{n}_1 \cdot \vec{n}_2 &= -1 \\ (\vec{p}_1 - \vec{p}_2) \cdot \vec{n}_2 &= 0. \end{aligned}$$

In this case the two faces are reported as a potential planar contact.

If F_1 is cylindric while F_2 is planar, we check whether the axis of the cylindric face is parallel to the planar surface, and that the distance between the axis and the plan is equal to the radius of the cylinder. Given \vec{a}_1 the unit vector in the direction of the axis of the cylindric face F_1 , r_1 its radius, \vec{n}_2 the normal of the planar face F_2 , \vec{p}_1 and \vec{p}_2 the respective position vectors of F_1 and F_2 , the above-mentioned check can be stated as follows;

$$\begin{aligned} \vec{a}_1 \cdot \vec{n}_2 &= 0 \\ (\vec{p}_1 - \vec{p}_2) \cdot \vec{n}_2 &= r_1. \end{aligned}$$

In this case the two faces are reported as a potential linear contact.

If both F_1 and F_2 are cylindric, we check whether the two axes coincide. Given \vec{a}_1 and \vec{a}_2 the unit vectors in the directions of axes of F_1 and F_2 respectively, and \vec{p}_1 and \vec{p}_2 their respective position vectors, the above-mentioned check can be stated as follows;

$$\begin{aligned} |\vec{a}_1 \cdot \vec{a}_2| &= 1 \\ |(\vec{p}_1 - \vec{p}_2) \cdot \vec{a}_2| &= 1. \end{aligned}$$

In this case, and if the radii are equal $r_1 = r_2$ the two faces are reported as potential cylindric contact. If the radii are not equal, a further test of surface orientation is done. In this case, if the cylinder with smaller diameter is oriented inwards, while the other one is oriented outwards, the solids of the two faces are reported as potentially in cylindric interference².

It is worth noticing that the above-mentioned criterion only allows for the detection of cylindric interferences for which the axes of the cylindric faces at both sides of the interaction coincide. Although, other configurations can be envisaged where the condition of axes coincidence is relaxed to simple parallelism. In the latter case, an additional condition on the perpendicular distance d between the two axes with respect to cylinder radii should be introduced. More precisely, the distance between axes should be less than

²The inverse case, when the cylinder with smaller diameter is oriented outwards and the other one is oriented inwards would denote a potential clearance. However, clearances are not considered in the scope of this work.

the sum of radii if partial, as well as complete, interferences are considered $d < r_1 + r_2$, and less than the difference of radii if only complete cylindric interferences are considered $d < |r_1 - r_2|$. The distance d can be calculated as follows;

$$d = \|(\vec{p}_1 - \vec{p}_2) - ((\vec{p}_1 - \vec{p}_2) \cdot \vec{a}_1) \vec{a}_1\|.$$

Such a configuration does not reflect an interpretable functional intention. Nonetheless, it may be encountered in industrial models as a result of imprecise geometric representation to what otherwise would be a coaxial configuration. Even if the detection of such configuration is technically possible with a minimal cost, it poses, however, a problem of interpretation. Such a configuration is thus not considered in the actual work, and is left for future extensions.

Combinations of other canonic faces such as spheres, cones and tori are also considered and studies, reporting candidates to circular and conic contacts.

Up to this stage, geometric interaction filtering is implemented in the class `BoundingBoxesGID` (for Bounding Boxes Geometric Interaction Detector) in our code. Note however that this class signal all potential candidates, as discussed above, leaving the final decision to boolean operators, as discussed in Section 5.3.

A major advantage of this technique is the order of magnitude drop in execution time it exhibits compared to mere boolean operators, as costly solid intersection calculation is avoided when not needed.

Furthermore, results obtained by this method readily contain geometric characteristics of the interaction, such as axes and normals, that are used to define a local Cartesian coordinate system, as shown in Section 6.3.1.

A drawback of this method is that it only detects interaction between surfaces that have a simple set of geometric characteristics; i.e. canonic surfaces. Contacts and interferences that involve free-form surfaces are not detected using this method, even if the opposite surface is canonic. However, in the context of this research, this is tolerated, and even favorable, given that we are only interested in functional surfaces, as shown in Section 5.1.

5.2.4 Local coordinate systems

CIs are shipped with local coordinate systems that are particularly important when assigning physical properties to these interfaces. As Chapter 6 will unfold, dynamic and kinematic behaviours of a CI are expressed with respect to these local coordinate systems.

The choice of a local coordinate system is thus not arbitrary. They are, in fact, orthogonal right-handed coordinate systems, conventionally defined³ in as much alignment as possible to geometric characteristics of the interface.

To this end, coordinate axes are defined based on normals and axes of symmetry whenever available. As an example, for a rectilinear contact (such as the one between a cylinder and a plane shown in Figure 5.4b), the z -axis is defined along the surface normal, while the x -axis is defined by the contact line, the y -axis is then deduced using the right-hand rule, as the vector product of the x - and the z -axes. For planar contacts (shown in Figure 5.4a), only the z -axis is deterministically defined, while the x - and y -axes are left to lie on the contact plane. The choice of the coordinate system origin is also of important significance. In our work, this point is chosen to be the barycenter of the interaction zone, whether it is a curve, a surface or a volume. Figure 5.4 shows more examples of CIs geometries associated with their conventional coordinate systems.

It is important to note that the choice of coordinate system is a matter of convention. What does really matter here is the coherence between conventions made at this stage, and those considered when assigning physical properties to the interface. For example, a simple planar contact delimits object translational motion along its normal, while leaving it free to translate parallel to its surface. By making a coherent choice of the local coordinate system, say the one shown on Figure 5.4, one can say that a planar contact eliminates object translational mobility *along the positive direction of the z -axis* of such a contact.

5.2.5 Conventional interface graph

The outcome of this phase is represented as a graph referred to as conventional interface graph (CIG). This is a mathematical model upon which we build our reasoning in phases to come.

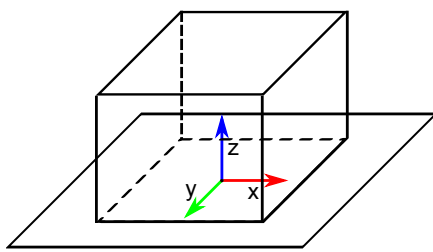
Definition 5.1 (Conventional interface graph). The CIG is a directed graph $G(\mathbb{C}, \mathbb{I})$ that has the set of all components in an assembly \mathbb{C} as its nodes, and the set of their CIs \mathbb{I} as its edges.

Initially, edges of the graph (i.e. CIs) contain information only about the geometric interaction between two nodes (i.e. two components), such as normals, axes, directions and radii, along with the interface local coordinate system transformation matrix with respect to the global coordinate system.

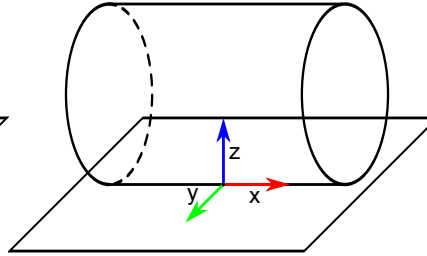
Even though some CIs are geometrically and functionally symmetric, such as those resulting from a simple planar contact, the concept of CI is still asymmetric in general.

A cylindric contact for instance generates a CI, we can clearly recognize the *outer* component from the *inner* one. This recognition can be made for many types of geometric configuration. CI asymmetry makes the CIG

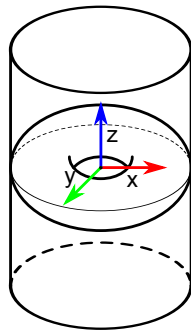
³Conventions are made in the scope of this work.



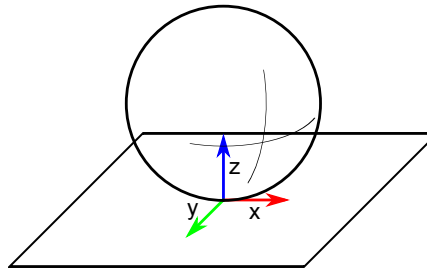
(a) Planar Contact



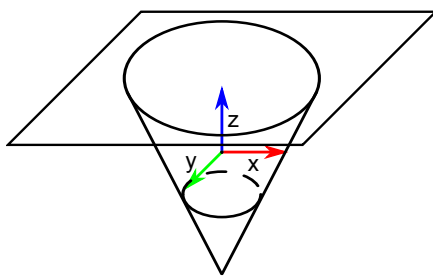
(b) Rectilinear Contact



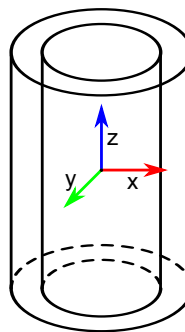
(c) Circular Contact



(d) Punctual Contact



(e) Conic Contact



(f) Cylindric Interference

Figure 5.4: A set of conventional interfaces with their associated coordinate systems.

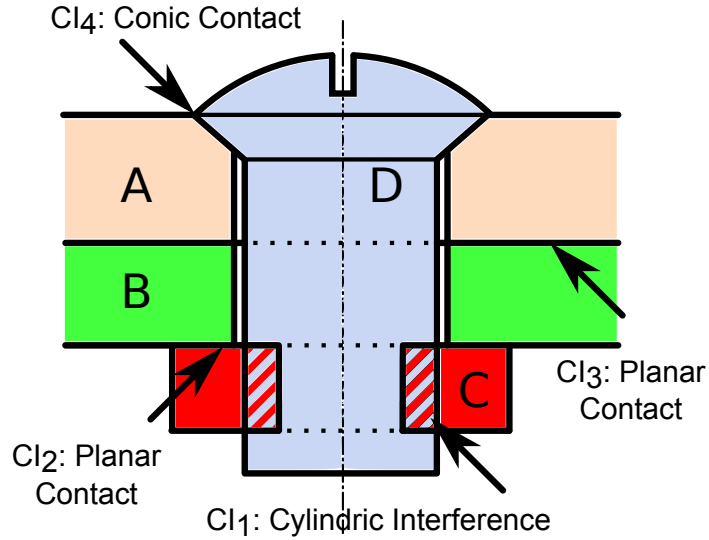


Figure 5.5: A cross-section in a partial geometric model model of a capscrew and a nut tightening up two plates, showing detected CIs.

a directed graph. Even when the geometric nature of the interface makes no distinction between the two components at each side, one of the two orientations is assumed to define the edge orientation in the CIG.

We refer to the graph produced by ignoring edge orientations in CIG as the conventional interface underling undirected graph (CIuG).

Figure 5.5 shows a cross-section in a model of two plates; *A* and *B*, tightened up together by means of a capscrew *D* and a nut *C*. It also shows CIs between components as represented in the DMU: a cylindric interference CI_1 between *C* and *D*, two planar contacts CI_2 and CI_3 , between *B* and *C* and *A* and *B*, respectively, and a conic contact CI_4 between *A* and *D*. This information is initially unavailable in the input model; which is the DMU. Figure 5.6 shows the corresponding CIG, where components *A*, *B*, *C*, and *D* form the nodes, CI_1 , CI_2 , CI_3 and CI_4 form the edges, after being detected.

While classes `Component` and `ConventionalInterface` represent components and CIs respectively, the CIG is represented by the class `Conventional-`

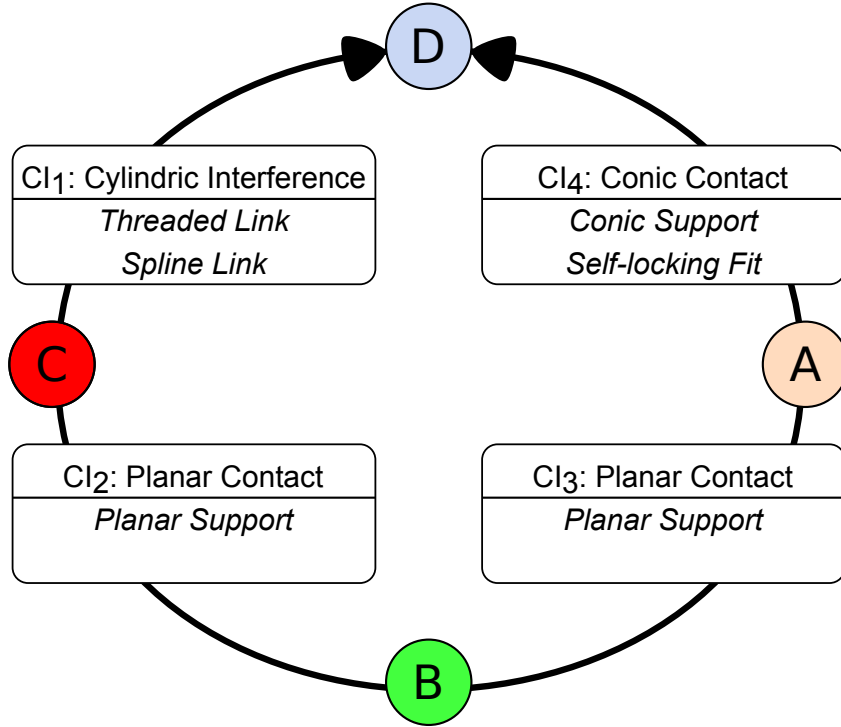


Figure 5.6: The CIG of the model represented in Figure 5.5, enriched with functional interpretations, as a set of FIs assigned to each CI. Only relevant edge orientations (those reflecting asymmetric geometric interface) are shown on the figure.

`InterfaceGraph` in our implementation.

5.3 Precise detection of interaction zones

The technique discussed in Section 5.2.3 is still approximate, it only filters out non-interacting canonic faces based on the comparison of the geometric parameters of their carrying surfaces. Faces that the previous phase reports are likely to interact, nonetheless, another measure is still needed to ensure real interaction. For example, two coaxial cylindric faces that share the same radius are reported as potential cylindric contact, though they may be afar along the axis and no actual contact takes place.

Moreover, the simple approach developed above fails short to deliver accurate interaction zones. Even though this shortage can be easily overlooked for the inference process, precise contact and interference zones are

still required for FEA purposes, as seen in Section 3.3.2.

A confirmation stage is hence necessary to validate candidates reported by earlier stages, and to produce precise interaction zones for those that are indeed valid.

To this end we use boolean operator applied to pairs of maximal surfaces in case of candidate contacts, and to solids in case of candidate interferences.

In spite of their high cost, as mentioned in Section 5.2.3, boolean operators are used to obtain precise geometric zone necessary for further FEA treatment, hence, their cost are justified for positive candidates. Since this is a final verification phase, only small number of false positives survive the previous filters compared to the total number of candidates, minimizing the wasted time computing intersection between non-interacting solids.

Candidates resulting from the canonical face comparison are thus validated using intersection operators, if the resulting shape is not empty, the interaction is considered valid, and a CI is created to represent it, adding an edge to the CIG. The newly created CI is associated to a local Cartesian coordinate system as shown in Section 6.3.1.

To enable the referencing of precise interaction zones and their annotation with functional semantics, before the model is passed to applications such as FEA, the geometric model of a component is restructured according to those interaction zone. This makes the functional breakdown mentioned in Section 4.2.3 possible as soon as the semantic annotations are available.

The OpenCascade software library [149] is used in our implementation to conduct boolean operators whenever needed. The class `BooleanBBOXID`, a subclass of `BoundingBoxID`, implements the final precise detection of interaction zone in our code, overriding the method `verifyInteractions(int, int)` that constantly returns `true` in its superclass.

5.4 Form-functionality mapping

Section 4.2.5 showed how functionality is satisfied as a direct result of geometric shape and physical material properties of components and their interfaces [118]. Given exact shape and material properties at both sides of an CI we can then deterministically deduce the functional role it plays in an assembly. However, this deduction is not readily possible in the DMU, because of the following reasons.

Imperfect geometric representation Section 3.2 showed that the DMU geometrically represents the product through its constituent sub-assemblies and components at different levels of details. That means that many form simplifications may take place, leading to loss of geometric information. Those simplifications are influenced by some sort of conventions, either internal to one company, or agreed upon for a specific 3D models provider or library as mentioned in Section 4.2.5.

Some simplification conventions may become a *de facto* standard, but there has never been a well-defined globally-recognized standardization of such conventions and notions, in contrary with many traditional 2D blueprints notions, as seen in .

Incomplete material information A DMU may or may not have a BOM attached to it (see Section 1.7). When a BOM is available, it provides an indication of physical properties that contributes to the functionality of a component or an interface. Once again, this information not reliable, neither in terms of existence (a DMU is not guaranteed to have a BOM), nor in terms of meaningfulness (there is no standards governing what information a BOM should contain, and in which format).

Because of this incomplete knowledge about shape and material properties in a DMU, the immediate deduction of functionality is not possible. However, assumptions can be made despite this incompleteness. Those assumption can then be reduced when the missing knowledge is reconstructed from existing one.

In this work we only consider pure geometry. We thus assume that the the BOM is unavailable, or uninterpretable, which is the worse case. Assumptions about materials physical properties such as stiffness and adherence are discussed in Section 6.2 when reference states are introduced.

5.4.1 Multiple functional interpretation

Considering geometric simplifications conventions observed in the industry, a limited number of assumptions can be made about the real shape when a specific geometric configuration is encountered in a DMU.

Section 4.2.5 showed that different FIs, such as threaded links and spline links can be represented as simple cylindric interference in a DMU. Another example of the use of the same geometric interface for different functional meanings is a simple cylindric contact that can refer either to a snug or to a loose fit (see Figure 3.3).

This allows us to link a given geometric interface, represented by a CI, to a set of *functional interpretations*, for each, an assumption is made about the real shape and physical properties of materials.

Definition 5.2 (Functional interpretation). A function interpretation is the assignment of one FI to a CI.

The assumption made for each interpretation provides the interface with a physical dimension, as developed in Section 6.3.1. More than one interpretation can be made per CI (see Figure 4.8), thus more than one independent physical and behavioral assumption.

5.5 Conclusions

Our methods takes a pure geometric representation of a product as a set of solids representing components. In this work we adopt an ISO standard that represents the geometric model as a STEP-File [10].

The geometric analysis consists of detecting the geometric interaction between solids, those are contacts and interferences defined in Section 4.2.5.

A preliminary approach would be to use geometric boolean operators, preceded by early elimination of negatives by means of bounding boxes checks.

We are particularly interested in those interaction occurring between faces that are likely to connect components with a functional bound. Observation shows that those faces happen to lie on canonic surfaces. This observation allows us to optimize our detection algorithm, adding a secondary, relatively fast, elimination filter right after the bounding boxes check.

Final validation and generation of interaction zones is still done by means of boolean operators, calculating intersection between potentially interacting elements.

This method is able to efficiently detect geometric interactions between canonic surfaces. The intermediate filtering between bounding boxes check and boolean operators validation reduces the detection time by orders of magnitude.

Geometric interactions of interest define CIs that link components together generating a directed graph called the CIG.

Once CIs are geometrically recognized, they are provided a physical dimension through functional interpretations. Functional interpretations are assumptions about the functional intent of the interface, made in the light of its geometric properties in the DMU. As a result, each CI is interpreted into one or more FI. This imprecision is due to the lack of reliable information about exact geometry and materials physical properties. Checking the validity of those assumptions allows for the elimination of irrelevant interpretations. The elimination process is to be discussed in the following chapter.

Chapter 6

Qualitative Behavioral Analysis of Components Functional Interactions

The previous chapter has shown that component FI assignment can lead to multiple solutions due to shape simplifications between their real and digital shapes. The corresponding assumptions have used shape – function dependencies. To reduce the number of FI per CI to one, which is the real configuration, the purpose is now to refer to behaviors to filter out FIs. Because the input data is a purely geometric representation of a DMU, it is proposed to set up a qualitative approach to describe behaviors where it is possible to take advantage of the somewhat precise representation of components while reasoning qualitatively with physical parameters. Because this approach is qualitative, it is applicable to a wide range of stages in a PDP, up to early design stages where the physical parameters related to FIs and, more generally, to components, are not yet available.

In this chapter, we demonstrate such an approach, defining referential behavioral states against which the validity of functional assumptions is checked to narrow down the number of possibilities to one functional interpretation per CI.

This chapter is organized as follows: the concept of reference state and energy preserving hypotheses in assembly joints are presented in Section 6.2, then the qualitative representation of joint forces and velocity properties is introduced in Section 6.3 through the concepts of qualitative wrench and twist screws. Finally, the reasoning scheme to select appropriate functions for each CI, which is based on static equilibrium and statically indeterminate reference states, is presented in Section 6.4 and Section 6.5.

6.1 Behavioral study to bind form to functionality

The previous chapter showed how to build the CIG that reflects our initial knowledge about the assembly through a graph structure. Even when enriched with functional interpretations, this knowledge still shows functional uncertainty.

Uncertainty in the knowledge base stems from the fact that some CIs hold more than one functional interpretation, thus, they cannot yet be mapped to a single FI. To reduce such uncertainty, additional rules and/or facts are to be taken into consideration, before being able to speculate about FDs and FCs of components and component groups.

Since the geometric analysis is not sufficient by itself to lead to a decisive functional resolution, as shown in Section 5.4, the concept of mechanical behavior, as discussed in Section 2.3.1, is borrowed here to take advantage of the form–function–behavior dependencies to strengthen the relationship between geometry and functionality so that efficient decisions can be taken over FIs to reduce them to one per CI, wherever needed is the assembly input.

Behavior can cover a wide spectrum of physical phenomena. In the scope of FE simulations, mechanical properties of components and their interactions, such as reciprocal forces between components taking place at each CI and force cycles tightening groups of components together are key elements that can be exploited to define behavioral models. Even though these mechanical properties are directly related to boundary conditions required by FE models, they are also of general interest. For example, reciprocal forces express whether a component can or cannot move with respect to its neighbors, which is also of interest in configurations of assembly or disassembly processes. Therefore, the concept of behavior and its qualitative approach can be seen as a generic tool that can be used to probe a DMU.

6.2 Reference states

Reference states (RSs) reflect rules that apply to the domain of discourse. In this sense, RSs are domain knowledge, as introduced in Section 2.6.1.

This knowledge may clarify ambiguity by reducing uncertainty in the knowledge base, e.g., reducing the number of functional interpretations per CI to ideally one. It also may produce certain new facts, such as the existence of a functional group that ties some subset of components. A RS is formally defined as follows.

Definition 6.1 (Reference state). A reference state (RS) is characterized by a set of input and a set of output physical parameters applied to a subset of components of a DMU. This subset may cover the whole DMU and the DMU is processed as given in its input geometric setting. A physical

behavior, consistent with the input and output parameters set up and a set of hypotheses is associated with the DMU to express the corresponding physical phenomenon. This behavior is characterized by a set of equations that takes as input the set of input parameters characterizing the state and the geometric configuration of the DMU and produces as output the set of physical parameters characterizing this state. A reference state can match any specific *state* of the product lifecycle, e.g., assembly process, working condition.

This definition straightforwardly relates to the notion of function (see Section 2.2) that can be related to sets of input and output parameters of a product or sub-system. Here, the purpose is to characterize a RS with its input parameters, behavior equations, hypotheses and study the output parameters that will characterize functions at some CIs of some components. From the characterized functions, it will be possible then to confront them to the assigned FIs and discard some of them when inconsistencies appear, i.e., the function expressed by a FI differs from the function derived from the output of the state behavior.

Indeed, the hypotheses and behavior equations are formed in terms of rules against which our knowledge about the DMU is checked, reducing uncertainty by refutation and producing facts by deduction. It has to be noticed that the principle of qualitative reasoning will explicitly, or implicitly, refer to relative physical values between components or products.

Different RSs have been recognized, both static and kinematic. Only static RSs, namely *static equilibrium* (see Section 6.4), and *static determinacy* (see Section 6.5), have been implemented in the scope of this work, at a first step, even though others are discussed such as *kinematic chains*. Initially, all studied RSs consider components as rigid bodies, unless rigidity proves to be impossible, that is, no possible functional interpretation satisfies this hypothesis.

Hypothesis 6.1 (Rigid bodies). Unless otherwise stated, components materials are assumed to be of high stiffness such that components are considered as rigid bodies, i.e., the loading conditions of a component do not alter its geometry. It is a common hypothesis for manufactured products where steel and other materials often lead to this hypothesis. It is also a common hypothesis used to define the boundary conditions of FE models prior to the use the deformation models expressed through the FE method.

This allows us to safely apply rigid body statics and kinematics throughout our analysis. This means also that the geometry of the DMU can be used straightforwardly.

Another common assumption that we make in default of any other clue is that connections between components of the assembly are ideal, thus frictionless. This can be generally formalized as follows.

Hypothesis 6.2 (Conservative systems). Unless otherwise stated, the set of components of the assembly forms an energy-conservative system. This hypothesis imply that contacts are frictionless, i.e., there is no tangential force between contact surfaces between any two components. We assume the state of contacts is not changing, i.e., a pair of solids having a planar support remain in contact. If not so, this means that the current RS is no valid and a switch to a new one must be operated.

This allows us to safely alternate between kinematic and static properties of interfaces, knowing that no work is done when two components move as per DoF allowed by an FI, or when internal loads propagate from one component to another through an FI. Again, this is the default configuration often used during design processes. Friction, however, can be of function importance and, in this particular case, lead to specific models as it will appear later on.

In the scope of this work, we only consider mechanical interactions between components in a product. This is because the recognition of other types of interactions, such as electromagnetic ones, require information beyond product geometry. Such information is not available in a DMU the way we consider it as an input to our analysis (see Section 1.11). Indeed, the objective of the proposed approach is to set up a process as automated as possible, in a first place. We state the aforementioned assumption as follows.

Hypothesis 6.3 (Mechanical interactions). In the scope of this study, we consider forces internal to a product to be purely mechanical and generated by solid components.

This allows us to ignore forces generated by electromagnetic fields, fluids, ... and to be consistent with the assembly model effectively input, i.e., the geometry of each component is known but there is no explicit representation of fluid and gaz domains in a DMU.

6.3 Qualitative representation of physical properties

In order to relate geometry to behavior, as a prerequisite to relate geometry to function, we need first to dress geometry in a DMU with physical properties. Properties of interest can either originate from statics, such as forces and torques¹ or kinematics, such as linear and angular velocities.

As long as functionality is concerned, this dressing should happen at the component interface level, i.e., the CI level, as it is the place from which

¹A torque being the moment of a force vector about a given axis.

functionality actually emerges. Section 5.4.1 showed how a functional interpretation of a CI connects it to a geometrically possible FI, in the light of industrial common practices. In fact, physical behavior, such as static and kinematic properties of an interface, directly relates to function [72, 136], as shown in Section 2.3.1. Making a functional assumption about a CI, the way a functional interpretation does, allows us to cast a behavioral – thus physical– dimension on the interface, as will be shown in more details hereafter.

Section 4.1 showed the motivation behind a purely qualitative approach. All studied physical attributes should thus support this requirement. In this section, we explain how static and kinematic properties can be expressed as qualitative values, before they are used to reason upon functional properties of components. We therefore provide our method with adequate data structures and their corresponding operations, arithmetics, and inference algorithms. Here, we promote an algorithmic approach since the concept of RS, the static and kinematic behaviors, are independent of a domain knowledge.

6.3.1 Qualitative physical dimension

An FI is given a physical dimension by associating it with a characteristic wrench screw² $W = \{\vec{f}|\vec{t}\}$ representing force and torque applied through the corresponding FI from one component onto the other one where FI is defined. This assumption is possible in light of Hypothesis 6.1 and Poinsot's theory [132] that states that any system of external force exerted on a rigid body can be resolved by one force, and a torque on a plane perpendicular to the force direction.

The abstract class **Screw** represents a general screw as two vectors (objects of the class **ScrewVector**) in our data structure scheme.

Values of force and torque vectors, however, are not scalars, as expected in a general dual vector, but qualitative symbols. These values decide in which direction, for a given vector component, a force or a torque may, or should, propagate. The following is a comprehensive list of nominal values that is used to this end.

Not Null: indicates that the underling FI propagates internal force/torque in either orientation along the corresponding axis.

Null: indicates that the underling FI does not propagate any internal force/torque along the corresponding axis.

Strictly Positive: indicates that the underling FI propagates internal force/torque, in the positive direction only, along the corresponding axis.

²Refer to Appendix C for details on screws and their representation as dual vectors.

Table 6.1: Qualitative vector values

Value	Symbol	Real interval
Not Null	\odot	$] - \infty, 0[\cup] 0, +\infty[$
Null	\odot	$[0, 0]$
Strictly Positive	\oplus	$] 0, +\infty[$
Strictly Negative	\ominus	$] - \infty, 0[$
Arbitrary	\otimes	$] - \infty, +\infty[$

Strictly Negative: indicates that the underling FI propagates internal force/torque, in the negative direction only, along the corresponding axis.

Arbitrary: indicates that the underling FI may or may not propagate internal force/torque, in either direction, along the corresponding axis.

Each FI is associated to a specific geometric configuration. For instance:

- spline and threaded links, and snug and loose fits, are associated geometrically with a couple of cylinders;
- planar and circular supports can be associated with a couple of planes and a plane and a torus, respectively;
- a linear support can be associated with a cylinder lying on a plane.

This enables an FI to acquire a local coordinate system, the same way a CI does (see Figure 5.4). A wrench screw, defined as a dual qualitative vector of values of Table 6.1, is then expressed in this local coordinate system and attached to the FI.

As a subclass of **Screw**, the class **RelativeScrew** represents qualitative dual vectors expressed in a local coordinate system in our application.

When a FI is processed, its local coordinate system is aligned with the local coordinate system of its underlying CI, which is in turn defined under the global coordinate system of the assembly. This allows the behavior model to locate the qualitative wrench screw globally with respect to the DMU.

Wrench and twist screws assigned to a planar support

When a *planar support*, an FI, is associated with a *planar contact*, a CI, through a functional interpretation, the corresponding interface refers to two components C_1 and C_2 . Let us consider that C_1 is chosen as reference component and CI is characterized by its imprint on its surface. Then, a

local coordinate system is assumed to have its z -axis coinciding with the contact plane normal, and its origin \vec{O} at the barycenter of CI, as shown in Figure 5.4a. C_1 being chosen as reference, we then observe that for an ideal, frictionless, planar support, as suggested by Hypothesis 6.2, all infinitesimal external forces that may be exerted by C_2 on C_1 through CI are normal to the plane, thus parallel to the z -axis of the interface local coordinate system. A force can then be expressed in the local coordinate system as $d\vec{f} = (0, 0, df_z)$ where $df_z < 0$ is the force per infinitesimal area of CI. Hence, the vector representing the sum of these infinitesimal forces is of the form:

$$\vec{f} = (0, 0, f_z) \text{ where } f_z < 0. \quad (6.1)$$

To compute the moment vector of \vec{f} about the origin $\vec{O} = (0, 0, 0)$, we note that $d\vec{f}$ produces a moment of the form: $d\vec{t} = (\vec{P} - \vec{O}) \times d\vec{f}$, where $\vec{P} = (x, y, 0)$ is the position vector where $d\vec{f}$ applies in CI, and \times is the vector cross product operator.

$$\begin{aligned} d\vec{t} &= (\vec{P} - \vec{O}) \times d\vec{f}, \\ &= (x, y, 0) \times (0, 0, df_z), \\ &= (y df_z, -x df_z, 0), \\ &= (dt_x, dt_y, 0) \end{aligned}$$

We observe then that the sum of these infinitesimal torques about O is of the form:

$$\vec{t}_{(x_0, y_0, z_0)} = (t_x, t_y, 0). \quad (6.2)$$

In fact, Equations 6.1 and 6.2 can be expressed by means of a wrench screw W_1 using nominal values of Table 6.1.

$$W_1 = \left\{ \begin{array}{c|c} \odot & \otimes \\ \odot & \otimes \\ \ominus & \odot \end{array} \right\} \quad (6.3)$$

This physical understanding of a functional interface leads to the same result if the phenomenon is studied from a pure kinematic perspective. In the case of *planar support*, we define FI by its twist screw T_1 that represents the relative velocity (v_x, v_y, v_z) and relative rotational speed (w_x, w_y, w_z) vectors between C_1 and C_2 as:

$$T_1 = \{\vec{w}|\vec{v}\} = \left\{ \begin{array}{c|c} 0 & v_x \\ 0 & v_y \\ w_z & 0 \end{array} \right\}. \quad (6.4)$$

This kinematic standpoint can be also regarded from a functional perspective since T_1 expresses the relative movements between C_1 and C_2 when their contact is preserved.

Definition C.2 states that a pair of wrench screw and twist screw is reciprocal when the virtual work of the wrench on the twist equals zero. This can be stated as follows:

$$\begin{aligned}
 dW &= 0 \\
 (W \odot T)dt &= 0 \\
 W \odot T &= 0 \\
 \{\vec{f}|\vec{m}\} \odot \{\vec{\omega}|\vec{v}\} &= 0 \\
 \vec{f} \cdot \vec{v} + \vec{m} \cdot \vec{\omega} &= 0.
 \end{aligned} \tag{6.5}$$

In the context of this thesis, the component interfaces conform to Hypothesis 6.2 by default. Hence, contacts are frictionless and are neither power generators nor consumers, meaning that the wrench and twist screws are reciprocal [133]³.

We thus obtain the following wrench screw of a *planar support*, that, along with twist screw T_1 , satisfies Equation 6.5:

$$W_1 = \left\{ \begin{array}{c|c} 0 & t_x \\ 0 & t_y \\ f_z & 0 \end{array} \right\} \tag{6.6}$$

Now considering the unilateral geometric configurations of FI, it is necessary to add the condition $f_z < 0$ to Equation 6.6. We then can express W_1 using qualitative values as shown in Equation 6.3.

Screws applied to cylindric contacts

Another example is a *loose fit* between C_1 and C_2 , which is associated to a *cylindric contact* through a functional interpretation. In this case, we define the local coordinate system to lay its z -axis on the cylinder axis, while the origin of coordinates \vec{O} coincides with the same axis at barycentric position of the CI.

Then, we observe that all infinitesimal external forces are radial, i.e., normal to the contact surface on C_1 , in an ideal, frictionless support, as suggested by Hypothesis 6.2. The vector representing the force applied by C_2 on C_1 over a cylindric support is then of the form:

$$d\vec{f} = (df_x, df_y, 0) \tag{6.7}$$

where the z -axis coincides with the cylinder axis.

The moment, at point O , of an infinitesimal force $d\vec{f} = (df_x, df_y, 0)$ is given by the equation $d\vec{t} = (\vec{P} - \vec{O}) \times d\vec{f}$, where $\vec{O} = (0, 0, 0)$ is the

³We actually consider a screw space here, i.e., a space of all screws generated by linear combinations of one screw of the given form, instead of a single screw, and its reciprocal screw space. We, however, stick to the term *screw* in the rest of the document for the sake of simplicity.

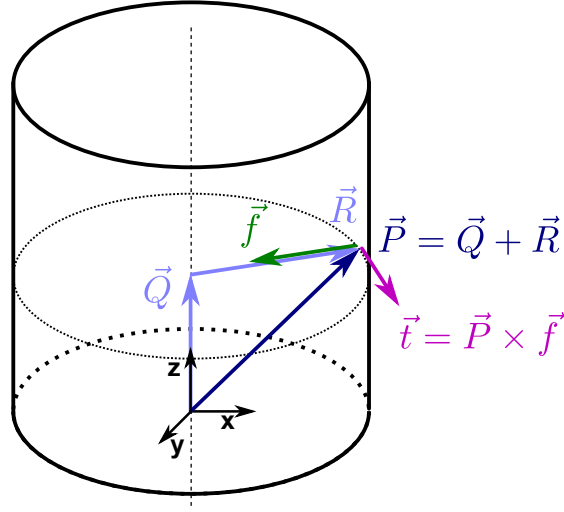


Figure 6.1: Vectors of position \vec{P} (axial, \vec{Q} , and radial, \vec{R} , components), force, \vec{f} , and torque \vec{t} of a cylindric support, represented in the local coordinate system of a cylindric contact.

origin of the local coordinate system in use, and \vec{P} is the position where the infinitesimal force applies. \vec{P} can be decomposed into two components; the axial component $\vec{Q} = (0, 0, z)$, and the radial component $\vec{R} = (x, y, 0)$, we note that $x \cdot df_y = y \cdot df_x$ since the infinitesimal force applies at \vec{P} (see Figure 6.1).

$$\begin{aligned}
 d\vec{t} &= (\vec{P} - \vec{O}) \times d\vec{f}, \\
 &= (\vec{Q} + \vec{R}) \times d\vec{f}, \\
 &= \vec{Q} \times d\vec{f} + \vec{R} \times d\vec{f}, \\
 &= (0, 0, z) \times (df_x, df_y, 0) + (x, y, 0) \times (df_x, df_y, 0), \\
 &= (0, 0, z) \times (df_x, df_y, 0) + \vec{0}, \\
 &= (dt_x, dt_y, 0).
 \end{aligned}$$

The sum of such infinitesimal torques about the origin O , is then of the form:

$$\vec{t}_{(x_0, y_0, z_0)} = (t_x, t_y, 0). \quad (6.8)$$

Again, Equations 6.7 and 6.8 can be qualitatively expressed by means of a wrench screw W_2 :

$$W_2 = \left\{ \begin{array}{c|c} \otimes & \otimes \\ \otimes & \otimes \\ \odot & \odot \end{array} \right\}. \quad (6.9)$$

From a kinematic standpoint, a loose fit can be defined by its twist screw as follows:

$$\mathbf{T}_2 = \left\{ \begin{array}{c|c} 0 & 0 \\ 0 & 0 \\ w_z & v_z \end{array} \right\}. \quad (6.10)$$

\mathbf{T}_2 can be also interpreted from a functional perspective where v_z and w_z are the output parameters that express the relative movements of C_2 with respect to C_1 .

Taking the reciprocal screw of \mathbf{T}_2 , we obtain the wrench screw of this FI:

$$\mathbf{W}_2 = \left\{ \begin{array}{c|c} f_x & t_x \\ f_y & t_y \\ 0 & 0 \end{array} \right\}. \quad (6.11)$$

Using qualitative values of Table 6.1, we obtain the same wrench screw as in Equation 6.9, showing that no other condition needs to be added to describe a *loose fit*.

Qualitative wrench screws applied to contacts

In an analogous manner, a qualitative wrench screw is assigned to each FI. Table 6.2 shows examples of FIs, and their associated qualitative wrench screws deduced using the same reasoning as mentioned above. Screws shown in the table are to be interpreted in the local coordinate systems of the corresponding CIs, as shown in Figure 5.4. Here, it is important to note that not all the FIs conform to Hypothesis 6.2. From a functional point of view, it is necessary to consider configurations where adherence between C_1 and C_2 plays a central role. This can be observed when comparing the *loose fit* and the *snug fit*. Independently, of the materials of C_1 and C_2 and of the diameter differences between C_1 and C_2 , the design principle of this fitting is to resist to components f_z and t_z under working conditions of a product. This justifies the content of \mathbf{W} and \mathbf{T} compared to the *loose fit*. This observation is critical since, the qualitative screws of the *snug fit* differ from that of the *loose fit* whereas their CI is identical, i.e., in the assembly model the conventional representations of these FIs are identical (see Chapters 4 and 5).

Though this is not detailed for sake of conciseness, a similar analysis applies to the *conical support* and *self-locking fit* where adherence is also of functional effect though it is combined, in this case, with the apex angle of the cone. This angle is a geometric parameter that influences the adherence effect of the conical fit. This could hinder the qualitative approach but the purpose is not to evaluate a design configuration, it is the analysis a valid ones. Consequently, a first approach can be the categorization of apex angles from a functional point of view. In this case, apex angles can be categorized

efficiently into *conical support* and *self-locking fit* using an angle threshold because this adherence phenomenon varies slowly with respect to the apex angle and is rather insensitive with respect to the categories of constitutive materials of C_1 and C_2 . This approach, however, is not the most generic one. A second one consists in dropping the reference to the apex angle, i.e., to geometry and refer to a specific physical behavior. This is the solution described here at Section 6.5.1.

6.3.2 Algebraic structure of qualitative values

To enable the use of qualitative screws against static equilibrium equations, certain arithmetic has to be defined on qualitative values represented in Table 6.1. Notably, the addition (+) and multiplication (.) operators. The semantics of these operators is defined through interval arithmetic [59], where each qualitative values represent real intervals as Table 6.1 shows.

Given K and $L \in [\mathbb{R}]$, a set of real intervals, an interval operator $\star : [\mathbb{R}]^2 \rightarrow [\mathbb{R}]$ is defined as the interval extension of the real operator $\star : \mathbb{R}^2 \rightarrow \mathbb{R}$ as follows [59]:

$$K \star L = \{z | \exists (x, y) \in K \times L, z = x \star y\}. \quad (6.12)$$

To this end, we define addition and multiplication operators on the set of qualitative values $Q = \{\odot, \oplus, \ominus, \odot, \otimes\}$ as an extension of real addition and multiplication, by replacing K and L in Equation 6.12 by the interval each value represents. We obtain the Cayley tables shown in Table 6.3.

Studying addition (+) in Table 6.3, we observe that:

- addition is closed on Q , i.e., all table cells are included in the entries;
- addition is associative, i.e., this can be established using Light's algorithm [97]);
- addition has an identity element \odot , i.e., its raw and column match the entries;
- addition is commutative, i.e., the table is symmetric.

We establish then, that $(Q, +)$ is a commutative monoid. Now, studying product (.) in Table 6.3, we observe that:

- product is closed on Q , i.e., all table cells are included in the entries;
- product is associative, i.e., this can be established using Light's algorithm;
- production is commutative, i.e., the table is symmetric.

Table 6.2: Different FIs and the CIs they originate from. Each FI is given a physical dimension using a qualitative wrench screw W and a qualitative twist screw T , defining its static and kinematic behaviors, respectively. Screws are expressed in the local coordinate system of the corresponding CI (see Figure 5.4).

FI	W	T	CI(s)
Threaded Link	$\begin{Bmatrix} \otimes & \otimes \\ \otimes & \otimes \\ \odot & \otimes \end{Bmatrix}$	$\begin{Bmatrix} \odot & \odot \\ \odot & \odot \\ \odot & \odot \end{Bmatrix}$	Cylindric Interference
Planar Support	$\begin{Bmatrix} \odot & \otimes \\ \odot & \otimes \\ \ominus & \odot \end{Bmatrix}$	$\begin{Bmatrix} \odot & \otimes \\ \odot & \otimes \\ \otimes & \oplus \end{Bmatrix}$	Planar Contact
Spline Link	$\begin{Bmatrix} \otimes & \otimes \\ \otimes & \otimes \\ \odot & \otimes \end{Bmatrix}$	$\begin{Bmatrix} \odot & \odot \\ \odot & \odot \\ \odot & \otimes \end{Bmatrix}$	Cylindric Interference
Snug Fit	$\begin{Bmatrix} \otimes & \otimes \\ \otimes & \otimes \\ \otimes & \otimes \end{Bmatrix}$	$\begin{Bmatrix} \odot & \odot \\ \odot & \odot \\ \odot & \odot \end{Bmatrix}$	Cylindric Contact
Loose Fit	$\begin{Bmatrix} \otimes & \otimes \\ \otimes & \otimes \\ \odot & \odot \end{Bmatrix}$	$\begin{Bmatrix} \odot & \odot \\ \odot & \odot \\ \otimes & \otimes \end{Bmatrix}$	Cylindric Contact
Linear Support	$\begin{Bmatrix} \odot & \otimes \\ \odot & \odot \\ \ominus & \odot \end{Bmatrix}$	$\begin{Bmatrix} \odot & \otimes \\ \otimes & \otimes \\ \otimes & \oplus \end{Bmatrix}$	Rectilinear Contact
Circular Lateral Support	$\begin{Bmatrix} \otimes & \odot \\ \otimes & \odot \\ \odot & \odot \end{Bmatrix}$	$\begin{Bmatrix} \otimes & \odot \\ \otimes & \odot \\ \otimes & \otimes \end{Bmatrix}$	Circular Contact
Circular Axial Support	$\begin{Bmatrix} \odot & \otimes \\ \odot & \otimes \\ \ominus & \odot \end{Bmatrix}$	$\begin{Bmatrix} \odot & \otimes \\ \odot & \otimes \\ \otimes & \oplus \end{Bmatrix}$	Circular Contact
Conical Support	$\begin{Bmatrix} \otimes & \otimes \\ \otimes & \otimes \\ \ominus & \odot \end{Bmatrix}$	$\begin{Bmatrix} \odot & \odot \\ \odot & \odot \\ \otimes & \oplus \end{Bmatrix}$	Conical Contact
Self-locking Fit	$\begin{Bmatrix} \otimes & \otimes \\ \otimes & \otimes \\ \otimes & \otimes \end{Bmatrix}$	$\begin{Bmatrix} \odot & \odot \\ \odot & \odot \\ \odot & \odot \end{Bmatrix}$	Conical Contact

Table 6.3: Addition (+) and production (.) Cayley tables over qualitative vector values.

+	⊙	⊕	⊖	⊙	⊗
⊙	⊙	⊕	⊖	⊙	⊗
⊕	⊕	⊕	⊗	⊗	⊗
⊖	⊖	⊗	⊖	⊗	⊗
⊙	⊙	⊗	⊗	⊗	⊗
⊗	⊗	⊗	⊗	⊗	⊗

.	⊙	⊕	⊖	⊙	⊗
⊙	⊙	⊙	⊙	⊙	⊙
⊕	⊙	⊕	⊖	⊙	⊗
⊖	⊙	⊖	⊕	⊙	⊗
⊙	⊙	⊙	⊙	⊙	⊗
⊗	⊙	⊗	⊗	⊗	⊗

We establish then that $(Q, .)$ is a commutative semi-group [98].

It can be shown, on a case-by-case basis, that product is distributive over addition, we thus infer that $(Q, +, .)$ is a semi-ring [98], which is commutative with \oplus as identity.

We also observe that:

$$\begin{aligned} \forall p, q \in Q, p+q = \odot &\implies p = \odot \wedge q = \odot; \\ \forall p, q \in Q, p \cdot q = \odot &\implies p = \odot \vee q = \odot. \end{aligned}$$

It follows that $(Q, +, .)$ is a *semi-field* [98].

Qualitative values and their arithmetic are defined through the class **ScrewValue** in our software application. The class also implements multiplication between real scalars and qualitative values using interval arithmetic again. A scalar x is replaced by its singleton interval equivalent $[x, x]$ in this case.

6.3.3 Coordinate systems alignment

Wrench and twist screws are expressed in the local coordinate systems of their corresponding FI by means of dual vectors which are referred to as *local dual vectors*.

Each local dual vector (an object of class **RelativeScrew** in our implementation) has its own Cartesian coordinate system in 3D. When dual vectors are added, subtracted or multiplied, they must be expressed in the same coordinate system, i.e., same reference frame and same origin. Unless all dual vectors share the same coordinate system, a coordinate-system alignment is thus necessary whenever dual vectors are summed.

Given a dual vector $S = \{\vec{v}|\vec{m}\}$ represented in a Cartesian coordinate system $(o, \vec{e}_x, \vec{e}_y, \vec{e}_z)$, we represent S as $\{\vec{v}'|\vec{m}'\}$ in a new Cartesian coordinate system $(o, \vec{e}_x', \vec{e}_y', \vec{e}_z')$. We note that:

$$\begin{aligned} \vec{v}' &= \Omega \times \vec{v}, \\ \vec{m}' &= \Omega \times \vec{m}, \end{aligned} \tag{6.13}$$

where Ω is the rotation matrix of \vec{v} and \vec{m} in the new coordinate system:

$$\Omega = \begin{bmatrix} \vec{e}_x' \cdot \vec{e}_x & \vec{e}_x' \cdot \vec{e}_y & \vec{e}_x' \cdot \vec{e}_z \\ \vec{e}_y' \cdot \vec{e}_x & \vec{e}_y' \cdot \vec{e}_y & \vec{e}_y' \cdot \vec{e}_z \\ \vec{e}_z' \cdot \vec{e}_x & \vec{e}_z' \cdot \vec{e}_y & \vec{e}_z' \cdot \vec{e}_z \end{bmatrix}.$$

If the two local coordinate systems share the same axes, but not the same origin, it means that the underlying wrench or twist screws do not share the same point of application. Given a dual vector $S = \{\vec{v}|\vec{m}\}$ represented in a Cartesian coordinate system $(o, \vec{e}_x, \vec{e}_y, \vec{e}_z)$, we represent S as $\{\vec{v}'|\vec{m}'\}$ in a new Cartesian coordinate system $(o', \vec{e}_x', \vec{e}_y', \vec{e}_z')$. We note that:

$$\begin{aligned} \vec{v}' &= \vec{v} \\ \vec{m}' &= \vec{m} + \vec{\lambda} \times \vec{v}, \end{aligned} \tag{6.14}$$

where $\vec{\lambda}$ is the translation vector starting at o' and ending at o , $\vec{\lambda} = \vec{o'o}$.

Now when neither axes are aligned nor the origin is shared between the new and the current Cartesian coordinate systems, we first apply Equation 6.13, then Equation 6.14 to obtain a dual vector expressed in the new coordinate system.

These operators are implemented by the method `changeBasis(gp_Ax2)` of the class `RelativeScrew` in our software.

6.4 Reference state I: Static equilibrium

Static mechanical equilibrium RS builds on the fundamental law of dynamics (Newton's second law), assuming that all components are at rest.

Hypothesis 6.4 (Static equilibrium). Components of an assembly are at static equilibrium.

This means that mechanical equilibrium equations hold on each component. More precisely, components in a DMU should not fall apart either when the assembly is at rest state or in working conditions. This is equivalent to consider that every component should not be free to move along any translational movement, which can allow it to fall apart. A component however, can rotate freely because a rotational movement cannot separate a component from its assembly. Because of these observations, this RS can be applied to a very wide range of DMUs.

Those equations extend our knowledge about the DMU that is built on its pure geometry model so far. They thus allow the qualitative analysis to proceed with farther reasoning, particularly, the reduction of the number of FIs. The mechanical system studied is reduced to a standalone component and its equilibrium is studied under the actions of its neighboring components onto it.

6.4.1 Static equilibrium equations

A rigid body, i.e., an assembly component, is at mechanical equilibrium if and only if it satisfies the following conditions:

1. The vector sum of all external forces applied to this rigid body is zero;
2. The vector sum of all external torques applied to this rigid body is zero around any given axis.

Since CIs encapsulate all geometric interactions of a component, all mechanical forces are applied through these interfaces. In the light of Hypothesis 6.3, we can thus state that all forces external to a component are applied through its CIs.

Let \mathbb{I}_C be the set of all CIs of a component C . Given \vec{f}_i the force vector and \vec{t}_i the torque vector that represent the resultants of external forces applied to C [132] through its i^{th} interface by its i^{th} neighboring component, $i \in \mathbb{I}_C$. \vec{f}_i and \vec{t}_i are the components of a wrench screw $W_i = \{\vec{f}_i | \vec{t}_i\}$ applied to C at its i^{th} CI, $i \in \mathbb{I}_C$, then the static mechanical equilibrium of C is obtained when all such wrench screws sum up to zero $\{\vec{0} | \vec{0}\}$:

$$\sum_{i \in \mathbb{I}_C} \{\vec{f}_i | \vec{t}_i\} = \{\vec{0} | \vec{0}\}. \quad (6.15)$$

Hence, the goal of the static equilibrium RS, is to check functional interpretations, FIs, associated to each CI against equation 6.15. This means that more than one wrench screw W_i , i.e., more than one FI, may exists per CI and it is the purpose of the behavior expressed by the static equilibrium RS to provide the qualitative analysis with output parameters that express the behavior of C with respect to this state. Checking a FI ends up verifying if C can reach a static equilibrium state.

Summation (addition or subtraction) can only take places on wrench screws sharing the same coordinate system. In our implementation, summation of two dual vectors generate a third one expressed under the local coordinate system of the first operands, where a copy of the second operand undergo a coordinate system alignment, as explained in Section 6.3.3, before being summed up. This functionality is implemented by methods `add(Screw)` and `subtract(Screw)` of the class `RelativeScrew`.

Scaling (multiplication by a scalar) is also defined on qualitative dual vectors by the method `scale()`, which accepts either a real `scale(float)` or a qualitative value `scale(ScrewValue)` as a parameter.

6.4.2 Graph search to eliminate irrelevant FIs

An exhaustive search algorithm that returns all valid solutions of a system, eliminating invalid functional interpretations for each CI, is presented in

this section. A valid solution is a set of interpretations, that is a set of FIs, that satisfy static mechanical equilibrium RS for a given CI. Algorithm 1 sketches the outlines of such an approach.

Algorithm 1 Mechanical analysis

Procedure: *analyze*

- for** each component c **do**
- mark c as OPEN
- end for**
- while** there is still a component marked as OPEN **do**
- $c \leftarrow \text{nextOpenComponent}()$
- $\text{initScrew} \leftarrow \{(\odot, \odot, \odot) | (\ominus, \ominus, \ominus)\}$
- mark all FIs of all CIs of c as invalid
- $\text{calculateSum}(c, 0, 0, \text{initScrew})$
- for** all invalid FIs: fi **do**
- $ci \leftarrow \text{CI of } fi$
- $other \leftarrow \text{opposite component of } ci$
- mark $other$ as OPEN
- drop fi from possible interpretations of ci
- end for**
- if** $Q(c)$ didn't change **then**
- mark c as CLOSED
- end if**
- end while**

Procedure: $\text{calculateSum}(c, \text{level}, i, \text{base})$

- if** $\text{base} = \{(\oplus, \oplus, \oplus) | (\otimes, \otimes, \otimes)\}$ **then**
- mark all visited FIs as valid
- mark all FIs yet to be visited from here as valid
- else**
- if** $\text{level} = |I_c^*|$ **then**
- if** $\text{isNullable}(\text{base}) = \text{true}$ **then**
- mark all visited interpretations as valid
- end if**
- else**
- $ci \leftarrow \text{level-th element of } I_c^*$
- for** $i = 0$ **to** number of interpretation of ci **do**
- $fi \leftarrow i\text{-th interpretation of } ci$
- $\text{screw} \leftarrow \text{base} + fi.\text{screw}$
- $\text{calculateSum}(c, \text{level} + 1, i, \text{screw})$
- end for**
- end if**
- end if**

This algorithm traverses the CIG through procedure *analyze()*, visiting

each node at least once, to study the component equilibrium against Equation 6.15. All nodes of the CIG are initially marked as **open**, i.e., they are still to be visited. Though the RS described at Section 6.4.1 applies to a unique component, Algorithm 1 traverses the entire assembly model because the RS is applied individually to each component of a DMU.

The function *nextOpenComponent()* returns the next component to visit. This is the **open** component with maximum *certainty*. The certainty of a component $c \in \mathbb{C}$ is defined as the reciprocal (multiplicative inverse) of the product of the number of interpretations over functionally-valid FIs of the component:

$$Q(c) = \left(\prod_{i \in \mathbb{I}_c^*} |i| \right)^{-1}, \quad (6.16)$$

where $|i|$ is the number of functional interpretations of the interface i , i.e., the number of associated FIs, and \mathbb{I}_c^* is the set of CIs involving c for which there exists at least one functional interpretation (see Section 6.4.3).

If two components happen to have the same certainty, the one with the smallest $|\mathbb{I}_c^*|$ is chosen. This heuristic picks components with higher entropy, thus higher potential to introduce new information, therefore enhancing algorithm convergence time.

For the sake of efficiency, certainty is initially calculated and stored in terms of its reciprocal for each component. Certainty is only updated component-wise when a functional interpretation reduction occurs to one of its CI. Comparing certainties reduces to comparing their stored reciprocal, that are integers.

Component equilibrium is studied through procedure *calculateSum()*. Before the call to this procedure, all possible functional interpretations are marked as **invalid**. Procedure *calculateSum()* marks an interpretation as **valid** if it participates to a solution that satisfies Equation 6.15, as will be explained shortly. After the call, all interpretations that are still **invalid** clearly contradict the mechanical equilibrium RS, thus they are eliminated.

If the call to *calculateSum()* leads to the elimination of at least one interpretation, not only the state of the current component is preserved as **open**, also the opposite component of the eliminated interpretation interface is marked as **open** as well, even if it was **closed** before. This is because the removal of one functional interpretation of a CI introduces new knowledge that may in turn allow for new conclusions if equilibrium equations are checked again against the involved component.

If no functional interpretation is removed, this means that further reasoning on the component is meaningless, since it will lead to the same result (unless this fact is changed, by reducing the number of interpretations again through the reasoning on a neighboring component, see the previous paragraph), and the component then is marked as **closed**.

The procedure *calculateSum()* traverses each FI of component CIs recursively, through a depth first graph search, where each CI represents a level in the search tree. The search combines solutions at each node and checks their validity against Equation 6.15 at leaf level. This is done through the accumulation of wrench screws, where the wrench screw of the currently visited FI is added to a sum. The sum is eventually checked whether it is *nullable* or not; that is, whether or not Equation 6.15 may hold true. A nullable qualitative dual vector has all its values in $Q_0 = \{\odot, \otimes\}$. As mentioned at the beginning of Section 6.4 when stating more precisely the concept of static equilibrium of components used to filter out some FIs, a component able to rotate only cannot fall apart. Consequently, its strict static equilibrium equations that end up with: $Q_0 = \{\otimes, \odot\}$ (see Equation 6.15) can be relaxed into a weaker form: $Q_0 = \{\odot, \ominus, \oplus, \odot, \otimes\}$, that authorizes the component to rotate but not to translate.

If the resulting wrench screw is indeed nullable, all visited functional interpretations are marked as **valid**. To enable this backtracking, a record of visited interpretations is kept. This bookkeeping is not mentioned in the outlines of Algorithm 1 for the sake of simplicity.

An enhancement is introduced to the algorithm by the early determination of valid solutions when the sum of wrench screws related to component c represents a rigid link, that is, it equals the qualitative dual vector $\{(\odot, \odot, \odot) | (\odot, \odot, \odot)\}$ stating that c cannot move with respect to its neighboring components. In this case, summation will always lead to the same wrench screw, which is indeed nullable. Therefore, the recursion is interrupted, and interpretations still to be unfolded from this point, besides those already visited, are marked as **valid**, as the algorithm shows. This enhancement is justified by the fact that a fair amount of components in an assembly are generating such rigid links, e.g., screws, nuts. Consequently, their specific processing speeds up the overall treatment of an assembly.

At each iteration in procedure *analyze()*, *calculateSum()* is called, after which either a component is closed or at least one functional interpretation is dropped. A component can only be reopened at the expense of one dropped functional interpretation. Since the number of components is bound in a DMU, and so is the number of functional interpretations, the algorithm is guaranteed to terminate in finite time, given that *calculateSum()* does. Note that the number of functional interpretations is only a theoretical upper bound to the number of iterations, since only a limited number of functional interpretations are actually dropped from a given CIG.

Procedure *calculateSum()* is a breadth-first traversal algorithm, that runs in $O(\prod_I |i|)$ time in the worst case. This complexity however, is significantly reduced by early determination of rigid links.

Algorithm 1 is implemented by the method *analyseInterpretations()* of the class *ExhaustiveMechanicalAnalyser*.

6.4.3 Local failure of functional interpretation

A component fails to be functionally interpreted when no valid functional solution is found by Algorithm 1 for that component under assumed hypotheses. This occurs either when friction is involved (Hypothesis 6.2 does not hold) or when the model is not consistent (Hypothesis 6.4 does not hold, and components are likely to fall apart). In either case, the corresponding component is signaled to the user as a potential inconsistency. A failure in case of inconsistency related to the friction hypothesis can effectively occur when components are touching each other along planar faces in the assembly model and no other information is available. Back to the concept of conventional representation, this can refer to configurations where the corresponding CI is not a FI of type *planar support* but should be interpreted as a *glued link* or a *weld*. Though this type of FI has not been mentioned in Chapter 4 and Chapter 5, DMUs often represent *welds* and *glued links* as contact configurations. Processing efficiently these configurations is left for future work. Similarly, if the failure reflects a assembly model inconsistency, this is not in the scope of the present approach and is part of future work.

As a result, all functional interpretations of all CIs of the component are dropped. The set \mathbb{I}_c^* reduces to the empty set in this case, and the uncertainty of the component is assumed zero.

To avoid a cascading collapse in which a failure to interpret component CIs propagates to neighboring components, resulting in a failure to interpret them as well, an so on across the assembly model, graceful failure measures are inserted into the algorithm. This is obtained by virtually dropping defective CIs, introducing $\mathbb{I}^* \subseteq \mathbb{I}$, the set of functionally-yet-valid CIs; that is the set of CIs for which at least one functional interpretation exists.

This fault tolerant approach prevents a local inconsistency from hindering reasoning elsewhere across the DMU.

6.4.4 Graph search example

In this section we show a simple example that demonstrates the above-mentioned algorithm. The same principles also apply on more complex and complete assembly models.

We build on the example of the capscrew-nut assembly shown in Figure 5.5. Only the CIG of the model is passed as input to Algorithm 1, enriched with functional interpretations as shown in Figure 5.6.

Since the model is partial, and for the sake of conciseness, we only demonstrate here the execution of the algorithm on one component; that is C . Initially, components C and D are **open**. We know that the algorithm picks component C first, as its certainty,

$$Q(C) = (|CI_1| \times |CI_2|)^{-1} = 1/(2 \times 1) = 1/2$$

is the highest one in the assembly. For example, the certainty of component D ,

$$G(D) = (|CI_1| \times |CI_4|)^{-1} = 1/(2 \times 2) = 1/4$$

is lower than that of C .

Figure 6.2 shows the complete search tree that procedure *analyze()* of Algorithm 1 traverses to find valid solutions. The figure shows that at each level of the tree; that is a functionally-yet-valid CI of the component, two things happen:

- Firstly, the search tree forks in as many branches as remaining functional interpretations of CI;
- Secondly, and for each branch, the wrench screw of the FI of the corresponding functional interpretation is added qualitatively to the wrench screw sum.

At the leaf level, and when no more CIs are to be visited, the wrench screw sum is checked for ‘nullifiability’. The example shows two paths, the first one led to the wrench screw $\{(\otimes, \otimes, \ominus) | (\otimes, \otimes, \otimes)\}$ produced by the FI *spline link*. This solution path is rejected because of the existence of the strictly negative qualitative value \ominus , that cannot be nullified, contradicting Hypothesis 6.4. From a technological point of view, this is consistent because replacing the screw thread by a spline link would let the component D fall apart, as well as C .

The second solution path led to the wrench screw $\{(\odot, \odot, \otimes) | (\otimes, \otimes, \otimes)\}$. This solution is acceptable, as all values of the wrench screw are nullable, satisfying the mechanical equilibrium RS. Consequently, all functional interpretations along the way to this valid solution are marked as **valid**. Namely, CI_1 as a *threaded link* and CI_2 as a *planar support*.

As the functional interpretation of CI_1 as a *spline link* did not appear in any valid solution, it remains marked as **invalid**, and thus is eliminated. Certainty of component C is then updated to its new value $Q(C) = 1$, which ends the algorithm in the present case.

6.5 Reference state II: Static determinacy

The first RS leads to the reduction of the number of functional interfaces, eliminating geometrically suggested solutions that are mechanically invalid. This is a first illustration of a qualitative behavior analysis that is coupled to geometry, i.e., the shape of CIs, and functions, i.e., the FIs associated with CIs, to exploit the dependencies between form – function – behavior and robustly enrich a DMU with functional information at the level of component interfaces.

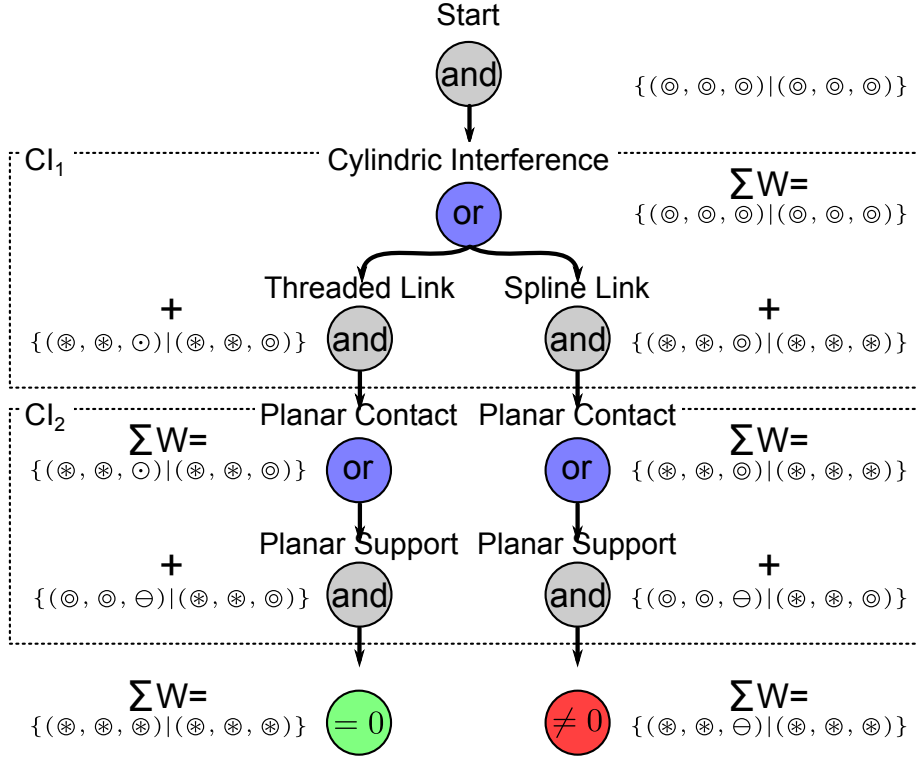


Figure 6.2: The search tree visited by Algorithm 1 in order to check the static equilibrium validity of FI of CI of component C shown in Figure 5.5. The tree shows two paths, one leading to a nullable qualitative dual vector, i.e., a valid solution, and the other one leads to a non-nullable qualitative dual vector, i.e., an invalid solution. This leads to the elimination of the function interpretation binding interface CI_1 with the FI *spline link*.

However, this elimination does not always lead to a one-to-one mapping between CIs and FIs, as for some configurations, more than one solution actually satisfies mechanical equilibrium, thus RS I.

To demonstrate such a configuration, we consider the example of a bolted assembly shown in Figure 5.5 with a conical head capscrew as component D on which we now focus. We recall that this functional knowledge about components, i.e., the ‘component names’, and component groups, i.e., the concept of bolted assembly, is not yet available at this stage of reasoning. Section 6.4.4 showed how Algorithm 1 runs on component C of this bolted assembly where C features a ‘planar support’ rather than a conical one, successfully reducing the number of valid solutions to one, thus setting the number of FI per CI to exactly one for C . Interfaces CI_1 and CI_2 are said to be definitively interpreted.

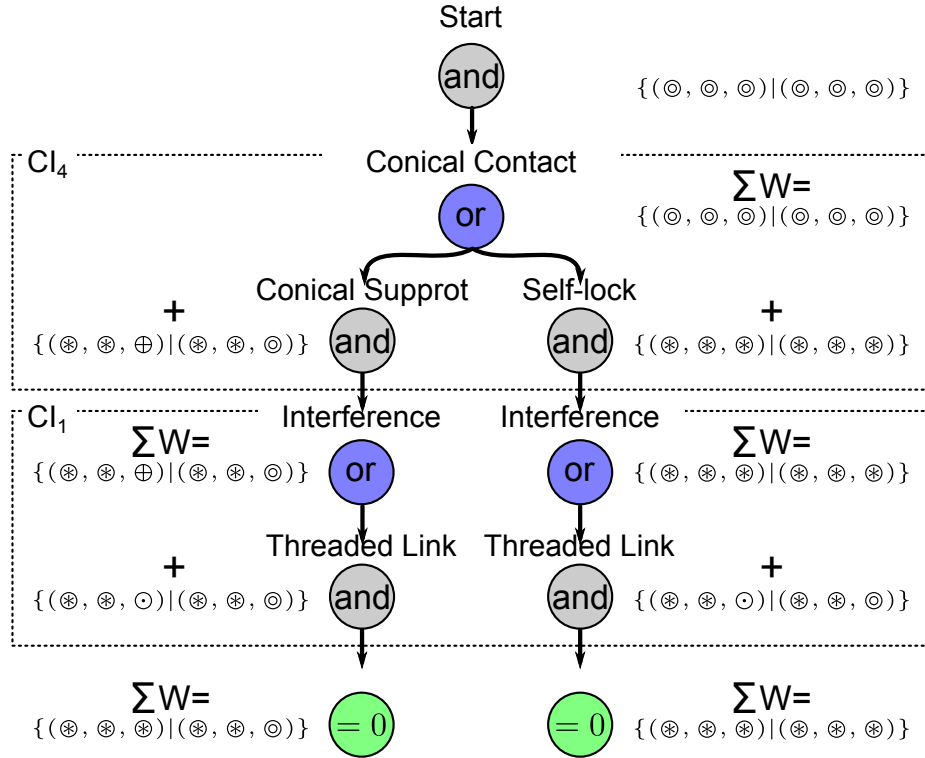


Figure 6.3: The search tree visited by Algorithm 1 in order to check the static validity of FIs of CIs of component D shown in Figure 5.5. The tree shows that all leaves, two in this case, leads to a nullable qualitative dual vector, thus a statically valid solution. No functional interpretation is eliminated in this case.

Now, considering the equilibrium of component D in accordance with Algorithm 1, we first note that only two solutions are left after the elimination of the interpretation of CI_1 as a spline link. The remaining solutions are:

- Interpreting CI_1 as a threaded link, and CI_4 as a *conical support*;
- Interpreting CI_1 as a threaded link, and CI_4 as a *self-locking link*.

As shown in Figure 6.3, both remaining solutions are indeed statically valid. The execution of procedure *analyze()* leads to no reduction in number of functional interpretations, thus no change in components uncertainty $H(D)$. Consequently, component D is closed, and will remain this way until the algorithm converges and returns.

This uncertainty prevents us from attributing functional properties to geometric elements in a decisive manner. The problem stems from the existence of more than one *valid* solution, even though solutions are not equally *relevant*.

A mechanism should thus be established to evaluate the relevance of a valid solution. If such a mechanism existed, discouraged, e.g., unnecessarily costly, solutions can be further rejected in favor of more convenient ones.

6.5.1 Statically indeterminate configurations

A close study of the example illustrated above shows that the two remaining solutions differ from each other in a discriminant way. Although both of them are statically valid, as the Algorithm 1 proves, the first solution is isostatic, or *statically determinate*, while the second one is hyperstatic, or *statically indeterminate* [123]. A statically indeterminate system can be characterized as follows (see Figure 6.4). This example shows a horizontal cantilevered beam at point A and its opposite extremity B leans on a rigid contact. Now, considering the static equilibrium equations of the beam that reduces to a simple planar problem, it comes, in the reference frame (A, \vec{x}, \vec{y}) :

$$f_{Ax} = 0, \quad (6.17)$$

$$f_{Ay} + f_{By} - F = 0, \quad (6.18)$$

$$t_A + Lf_{By} - \frac{L}{2}F = 0, \quad (6.19)$$

where the wrench screw at point A is defined by $\{f_{Ax}, f_{Ay}|t_A\}$ and with $\{0, f_{By}|0\}$ at point B , L is the length of the beam, and F is an external force.

It appears that this equation system is under determined, i.e., there are three unknowns and two equations. From a mechanical point of view, this system is said statically indeterminate of order 1. Such a configuration is classical in strength of materials and the approach commonly used to find a solution is to refer to the deformation behavior of the beam. Indeed, the bending behavior of the beam brings one more independent equation that can be used to solve the system. From a mechanical point of view, it means that the internal energy of the beam, i.e., its strain energy, contributes to the wrench screws at the interfaces A and B of the beam.

Now, coming back to the example of Figure 5.5 and the two alternatives of FIs that can be assigned to the conical contact of D , they differ from each other by the adherence effect that takes place with the *self-locking fit* when the cone apex angle is small (see Section 6.3.1). In this case, the adherence and compression of the conical component is able to develop an axial force, opposite to an external one, hence the designation *self-locking fit*.

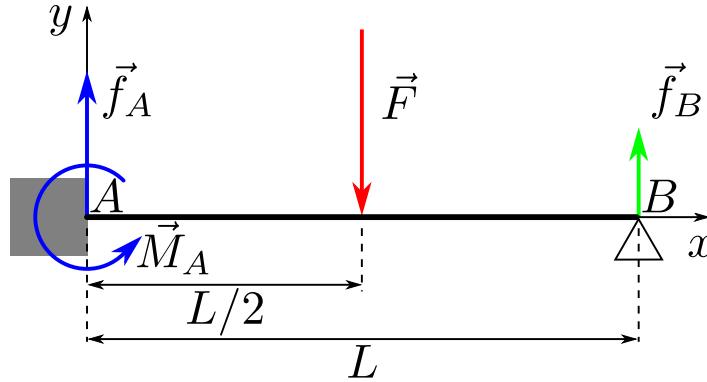


Figure 6.4: An example of statically indeterminate mechanical system with one parameter.

In the first solution of component D , i.e., *conical support*, and in the only solution of component C , each and every interface participates to the equilibrium of D . Dropping one interface would render the component invalid statically since the *conical support* does not incorporate any friction effect. This makes these solutions statically minimal, or determinate. In contrast, the second solution of component D , i.e., *self-locking fit*, is redundant in the sense that not every interface is strictly necessary to the equilibrium. In fact, a *self-locking link* is enough on its own to maintain the component D along the z -axis. Subsequently, a threaded link acting along the same axis with the same orientation as the *self-locking fit* increases the internal energy of this simple mechanical system while not being strictly necessary to the static equilibrium of D . This makes this second solution statically redundant, or indeterminate. Geometrically, this second solution is only valid for small cone apex angles.

Because of this redundancy, statically indeterminate configurations are avoidably expensive. They are thus uncommon in industrial products if not strictly necessary, e.g., in Figure 6.4, the rigid contact added at point B reduces the displacement of the beam at the point where F is applied compared to the displacement at the same point if the beam was cantilevered only. Here, adding the rigid contact increases the stiffness of the beam but adding this contact requires more components than the solution with a simple cantilevered beam, hence this solution is more complex to set up. However, a designer may resort to deliberately introduce statically indeterminate mechanism to convey certain functionality, such as reinforced fastening. This leads to the following hypothesis.

Hypothesis 6.5 (Static determinacy). Unless functionally justified, statically indeterminate structures are disfavored in a product assembly.

Consequently, the *self-locking fit* of Figure 5.5 is filtered out. It has to be pointed out that this filtering criterion is generic, hence it is independent of any threshold value, e.g., the cone apex angle threshold mentioned at Section 6.3.1.

More generally, Figure 6.5 shows examples of two plates assembled together by means of fasteners. Their assembly can be achieved through statically determinate structures, using one or two threaded links (see Figure 6.5a, b). Fastening can be secured through a statically indeterminate configuration with an additional locking nut, where static indeterminacy plays a functional role (see Figure 6.5c). Finally, Figure 6.5d shows a non-functional statically indeterminate configuration, if the *cylindric contact* is to be interpreted as a *snug fit*. The latter interpretation exhibits functional inconsistency and should be avoided in industrial products.

Hypothesis 6.5 provides the necessary criterion against which statically valid solutions that survived RS I can be filtered out.

In fact, statically indeterminate configurations could have been recognized during the evaluation of mechanical equilibrium in Algorithm 1. For example, a statically indeterminate solution could have been identified while wrench screws are summed, when a nullable qualitative value, i.e., \odot or \otimes , is added to another qualitative value rather than zero \odot . Accordingly, undesired statically indeterminate solution could have been eliminated since the first round.

However, some of statically indeterminate configurations constitute the only possible static solution of a component. In this case, the static indeterminate configuration is functional. Such a solution should not be eliminated when it is unique, however, whenever there is a functional interpretation conflict, a statically determinate solution prevails.

Consequently, statically indeterminate solution necessity cannot be judged component-wise. Since if a statically indeterminate solution is found unnecessary, as it has a statically determinate alternative for a given component, all functional interpretations participating to the solution would not be validated, thus are likely to be dropped (unless they participate to one of the statically determinate solutions). This elimination, however, may affect the propagation of equilibrium on neighboring components.

The static validity of model should thus be checked independently from its static determinacy. For this reason, the ‘pruning’ of the solution tree must occur after reasoning on the RS I is done, where all components playing a role in a statically indeterminate structure are studied at once. If a solution is to be eliminated, another statically determinate solution should be ensured for the whole structure. This analysis shows that RS II can be regarded as a reference state independent of RS I and must be applied after RS I.

This requires the study of internal forces propagation paths. In fact, statically indeterminate structures are characterized by a cyclic force propagation along one or more axis, e.g. in case of the example in Figure 6.4,

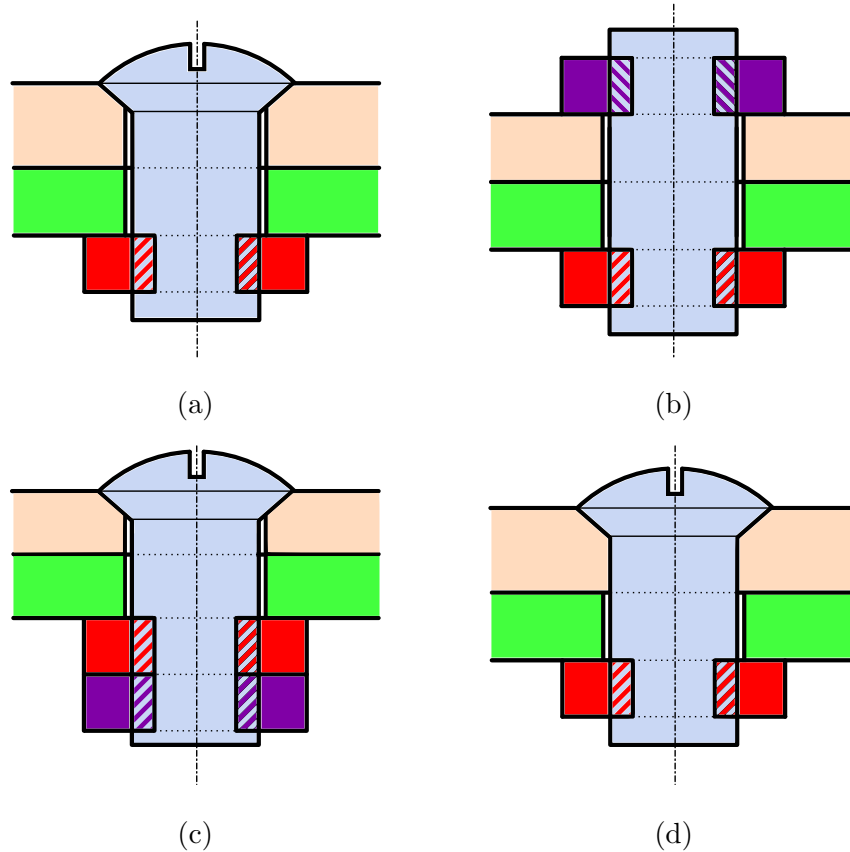


Figure 6.5: An assembly of two plates by means of a fastener: (a) A statically determinate solution, using a cap screw and a nut, with one threaded link, thus one internal load generator, (b) Another statically determinate solution, using a bolt with two nuts, and two threaded links, thus two internal load generators, (c) A statically indeterminate solution, with one cap screw and two nuts, the lowest one acting as locking nut. Static indeterminacy plays a functional role here, which is to reinforce fastening, (d) A statically indeterminate configuration if the cylindric interface between the cap screw and the upper plate (light brown) is to be interpreted as a snug fit. Static indeterminacy is non-functional here, and is considered as an unnecessary burden.

the statically indeterminate configuration is characterized by the load cycle along the \vec{y} axis. In order to outline indeterminate configurations, internal force propagation cycles should first be reported.

6.5.2 Force propagation and force propagation graphs

Internal force propagation paths can be formalized through force propagation graphs (FPGs), which are defined as follows.

Definition 6.2 (Force Propagation Graph). A force propagation graph (FPG) is a graph $\Gamma(D, J)$ that represents the propagation of internal forces in a model, along a given direction.

We know that each FPG is a subgraph of the CIuG, augmented with an orientation at each edge according to the conventional positive orientation of the force propagation direction taken as reference. This is a result of Hypothesis 6.3 that implies that internal forces only propagate through CIs, which are CIuG edges. Edge orientation is a matter of convention in both CIG and FPG. Orientation convention in the CIG (see Section 5.2.5) becomes meaningless when applied to force propagation, therefore we refer to the CIuG, the undirected version of CIG, as the super undirected graph of all FPGs.

In a given assembly model, there are as many FPGs as there are force propagation directions defined by FIs. Theoretically, a force can propagate in an assembly in all directions, that is, an infinite number of propagation directions in 3D space. This is justified when considering that components are behaving like deformable media, producing stress and strain fields when subjected to external forces. This leads to an infinite number of potential force propagation graphs. Here, we refer to Hypothesis 6.1 where components are rigid bodies. Consequently, forces follow the directions given by FIs and because the number of edges in CIuG is bound, there exists an upper limit on the number of its subgraphs, thus an upper limit on the number of FPGs.

Figure 6.6a gives an example of such a load cycle. This load cycle is initiated by an FI of type *threaded link* connecting C_1 and C_4 . The direction assigned to the force cycle is prescribed by the force of this *threaded link*. This force is then propagated with static equilibrium equations in that direction. Considering C_1 as reference component, its equilibrium equation determines the force applied by C_2 (light brown downward on Figure 6.6a). Then, moving to component C_2 , its equilibrium equation determines the force applied by C_3 (green upward on Figure 6.6a). Then, moving to component C_3 , its equilibrium equation determines the force applied by C_4 (red upward on Figure 6.6a). Finally, moving to component C_4 , its equilibrium equation determines the force applied by C_1 (gray upward on Figure 6.6a).

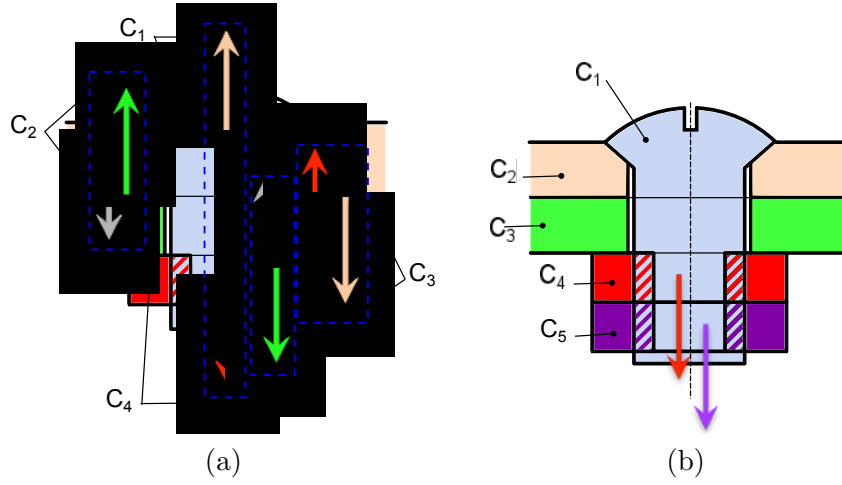


Figure 6.6: (a) An example of load cycle. C_i designates the components involved in the load cycle. C_1 contains a threaded link that originates the load cycle. The dotted rectangles represent the external forces applied to the corresponding component C_i . (b) An example of load cycle incorporating a statically indeterminate configuration. Here, the component C_1 is subjected to multiple forces with the same orientation that characterizes its static indeterminacy.

that is consistent, i.e., opposite, with the force applied by C_4 on C_1 that has been used initially. This ends the determination of the load cycle.

6.6 Reference state III: Assembly joint with threaded link as functional cluster

As Chapter 7 will reveal, later stages of our approach build on the organization of sets of components that participate to the fulfillment of one functionality in one functional group, and on the labeling of such groups by their corresponding FCs (see Section 4.2.4). An example is the clustering of components that are held up together using a tightening mechanism. The corresponding cluster defines effectively a functional group where the function can be stated as *fastening a set of components*, i.e., the tightened components are a subset of the cluster and the complementary subset describes the fasteners contributing to this function. If such a fastening process is accomplished through a threaded link, the functional group can be labeled as *assembly joint with threaded link* where assembly joint with threaded link is a FC designating a set of components held –or joint– together by means of a threaded component.

In fact, the detection of cyclic internal force propagation, required for

static determinacy analysis, regroups components that participate to common functions such as fastening. This enables their clustering into functional groups of *assembly joint with threaded link*.

Effectively, the set of components refers to a function. One component, the fastener, generates a given input parameter, i.e., here it is a force, and the interaction forces between components propagate through the assembly. The output parameters are characterized by the corresponding force propagation cycles and the force parameters at each CI of the cycle. This analysis enables the definition of a reference state expressing a fastening process. This is designated as RS III since the behavior observed is based on the action of a fastener and this action is effectively independent of the criteria governing RS I and II.

Hypothesis 6.6 (Force propagation). An FI capable of a fastening action generates a force through its neighboring components that propagates through the assembly and ends up producing a cyclic graph.

In case the force propagation mechanism does not produce a cycle it means that a fastening process may exist but:

- it takes place with objects outside the product, e.g., a vice has a fastening function but this function is performed on an object external to the product;
- or it indicates an inconsistent design.

In both cases, a user input is mandatory to characterize these configurations and they fall out of the scope of the proposed approach which concentrates, in a first place, on the assembly as a standalone set of components.

To achieve the above-mentioned benefits, an algorithm has to be established to detect cyclic internal force propagation paths.

6.6.1 Detection of force propagation cycles

In this section we define an algorithm that integrate Hypothesis 6.5 into our qualitative reasoning to further reduce the number of function interpretations per CI by the elimination of statically indeterminate solutions, whenever another statically determinate solution exists for the studied structure. As mentioned in the previous section, force propagation cycles are also useful to identify functional clusters of type *assembly joint with threaded link*. This algorithm will be used also in these configurations, whenever a threaded link is involved in such a cycle.

We are interested in the type of statically indeterminate structures that involve a FI generating forces internal to an assembly. We refer to those interfaces as *internal load generators*. Examples of such FIs are *threaded links* and *self-locking links*. Figure 6.6b gives an example configuration where

the load cycle produces a statically indeterminate loading of component C_1 . As observed, components C_4 and C_5 each apply a force on C_1 in the same direction and with the same orientation. This results in a statically indeterminate configuration of C_1 . Now, the criterion to identify a statically indeterminate configuration can be stated as follows. A component is subjected to statically indeterminate loading in the direction set for the load propagation process if there exists at least two forces applied to this component that share the same orientation.

FPGs are not readily separable from their supergraph; the CIuG. This is because the CIG, thus the CIuG, has no notion of force propagation direction expressed in a global coordinate system. Although this can be calculated based on CIs' associated local coordinate systems and their transformation matrices in the assembly coordinate system, this exhaustive subgraph generation is unnecessarily costly.

Alternatively, relevant force propagation subgraphs are generated incrementally, following edges of the CIuG, and starting with an edge that is indeed an internal load generator.

Procedure *findLoadCycles()* iterate over all interfaces that qualify as load generators, and identify the force cycle to which they participate by a call to function *followLoad()*. If the original load generator is actually a threaded link, as decided by function *threadedLink()*, the identified cycle is marked as functional group, and labeled as an *assembly joint with threaded link*.

Function *followLoad()* uses a classical breadth-first cycle detection algorithm. The initial call of the function takes an interface i which is guaranteed to be a load generator by the calling procedure. It also takes the force propagation direction and orientation, represented by its unit vector $\vec{\chi}$, which is an intrinsic property of each load generator. The algorithm applies to the FPG characterized by the direction $\vec{\chi}$, we refer to this graph as $\Gamma_{\vec{\chi}}$.

Since FPGs are not generated ahead of time, function *findReciprocal()* is used to 'sense' the CIG edges, and follow those who actually belong to $\Gamma_{\vec{\chi}}$. Function *findReciprocal()* is also responsible of the elimination of statically indeterminate functional interpretations. Static determinacy is evaluated each time the function is called, according to the direction $\vec{\chi}$. If at least one functional interpretation maintain a statically determinate solution, other static indeterminate functional interpretations are dropped.

Once a cyclic path is detected, the generation of the subgraph; i.e., $\Gamma_{\vec{\chi}}$, is stopped, and a candidate functional group is reported to procedure *findLoadCycles()*. The breadth first algorithm guarantees that the smallest cycle is detected for a given internal load generator.

Cases of statically indeterminate configurations are essentially characterized when the number of cumulative forces with the same orientation exceeds one. More general configurations are left for future work and reveal some design inconsistencies that are not in the scope of the present approach.

Algorithm 2 Load Cycles Detection

Procedure: *findLoadCycles**boltedJoints* $\leftarrow \emptyset$ **for all** load generating interfaces *i* **do**

mark all components of the assembly as WHITE.

cycle $\leftarrow \text{followLoad}(i, \vec{\chi})$ **if** *threadedLink*(*i*) **then***boltedJoints* $\leftarrow \text{boltedJoints} + \text{cycle}$ **end if****end for****Function:** *followLoad*(*i*, $\vec{\chi}$)*u* $\leftarrow \text{right}(i)$ **while true do**mark *u* as GRAY**for all** *j* in *findReciprocal*(*i*, *u*, $\vec{\chi}$) **do***v* $\leftarrow \text{opposite}(u, j)$ **if** *v* is GRAY **then***lc* \leftarrow new empty load cycle**while** *v* $\neq u$ **do**add *v* to *lc**backEdge* $\leftarrow \text{pred}[v]$ *v* $\leftarrow \text{opposite}(v, \text{backEdge})$ **end while****return** *lc***else if** *v* is WHITE **then***pred*[*v*] $\leftarrow j$ *followLoad*(*j*, $\vec{\chi}$)**end if****end for**mark *u* as BLACK**end while**

6.7 Conclusions

In this chapter, concrete methods and data structures are presented to capture mechanical behaviors at the interface, the component, and the component group levels. Each mechanical behavior is then employed to attribute functional knowledge to geometric elements at each of these levels, in relation to what was established in the literature, and studied in Section 2.3.

To enable this employment, referential behavioral standards are formalized in terms of RSs, defined in Definition 6.1. RSs add up to the knowledge base through the formalization of domain knowledge into hypotheses that are assumed to hold true in the context of a given state of the product (see Section 6.2).

A qualitative behavioral framework based on nominal physical values is defined in Section 6.3, to empower the purely qualitative approach promoted in Section 4.1. This framework casts a physical dimension onto different FIs in relation with the geometric configurations they interpret, as represented by their associated CIs. This physical reading of the model allows qualitative simulation processes to assess the model mechanical and functional validity.

Before any RS is applied to the model, the functional knowledge consists in what could be induced on pure geometrical basis; that is a number of functional associations, in terms of FIs, to each CI, as shown in Section 5.4.1. However, this number is not always one per interface. The multiple nature of functional interpretations in the general case introduces uncertainty to the knowledge base. This half-knowledge needs to be cleared if any reliable functional conclusion is to be withdrawn to the benefit of applications such as FE simulation and analysis.

To this end, Section 6.4 presented RS I, that is used to check knowledge consistency against static equilibrium equations. This leads to the reduction of uncertainty as statically invalid functional associations of CIs are dropped. However, examples show that although all statically invalid solutions are selected out, the number of those remaining; i.e., statically valid, still does not allow for a single positive functional solution.

RS II presented in Section 6.5 provides the qualitative approach with a supplementary criterion that takes into account not only solution validity, but also its quality with respect to its complexity, therefore its cost. Solution complexity is determined in view of its static determinacy. Unless static indeterminacy is functional in a solution, it is considered to be unrealistic in an industrial context, consequently such a solution is eliminated, leading to a further decrease in the number of FIs per CI.

RS III presented in Section 6.6 adds also another criterion that relates to the fastening function of a group of components. This function interacts with RS II through the definition of load cycles.

Most importantly, the qualitative approach set up has shown how it combines with the geometry of CIs to enforce functional information at the

level of FIs and extend to the components and to groups of components.

Finally, if the number of remaining statically determinate interpretations per CI still exceeds one, the most ‘probable’ solution would become a criterion. Probability here is defined by means of solution popularity in industrial DMUs when more than one survive RS I and RS II. Note that this measure is only taken as a last resort, in order to force a single FI per CI. In the studied industrial examples, however, this last filtering was not needed, hence not used and left for future work like other configurations where user’s interactions would be needed to process design inconsistencies or functions involving objects outside the DMUs.

The qualitative approach visited in this chapter allowed for the cleansing of the function knowledge associated to a DMU, reducing it to only positive facts about component interfaces and component groups. These facts actually provide the seeds of more elaborate knowledge, if combined with inference rules, as the following chapter will develop.

Chapter 7

Rule-based Reasoning to Derive Functional Denominations

As for now, only functional interactions were addressed in our qualitative analysis to filter out FIs and produce a precise spatial distribution of FIs. These FIs are consistent with the DMU both from some behavior point of view as well as geometrically. Now, the DMU is functionally interpreted in a definitive manner at the levels of functional interfaces and functional groups to take advantage of its form - behavior consistency and derive a consistent function at the level of each of its components.

However, in order to provide a complete functional description of a model, functionality at the functional unit level should be addressed. This chapter deals with annotating components with their functional denominations, i.e., their FDs.

Section 7.1 first illustrates the importance of such functional information, before Section 7.2 shows that this knowledge is the extension of the knowledge acquired so far after the application of domain specific rules, i.e., it takes advantage of the function dependency with respect to behavior and form concepts. Section 7.3 discusses alternative solutions to integrate such domain rules into our knowledge base. As a strongly related topic, knowledge representation is addressed in Section 7.4, before that Section 7.5 gives an in-depth demonstration of the formal reasoning process. Section 7.6 concludes, showing how functional knowledge is saturated generating the required functionally interpreted DMU model.

7.1 Knowledge at the functional unit level

The previous chapter showed a qualitative method that used algorithms, such as graph searches, to extend knowledge or reduce uncertainty about functional properties of an assembly. This method came out with functional facts, particularly at the levels of component interactions, that is, at the functional interface and the functional group levels, in a reference to what was outlined in Section 3.4. The corresponding algorithms mapped a consistent and qualitative behavior to functional interfaces, uniquely associating each CI to one FI. As an example of such behavior, this mapping also permitted the clustering of components tightened up together by means of a threaded link into functional groups referred to as bolted joints. In addition to the CIG, each behavior extends the DMU structure with a spatial mapping of the behavior functional meaning.

This algorithmic inference enriched our knowledge about the DMU in two ways:

- First, reducing ambiguity in statements such as *interface i is either a threaded link or a spline link*. This is done through the elimination of invalid, and unsuitable alternatives (see Section 6.4). For example, interface CI_1 in Figure 5.5 is definitively identified as *threaded link*.
- Introducing new facts such as stating that *interface i participates to an assembly joint with threaded link*. This is done through the detection of internal load cycles that tighten up components together (see Section 6.5). For example, interfaces CI_1 , CI_2 , CI_3 and CI_4 in Figure 5.5 are found to take part to the same *assembly joint with threaded link*.

In this sense, our knowledge base only contains positive facts after the application of RSs validation. The functional interpretations of geometric configurations are now unique, and uncertainty is reduced to zero. However, the functional knowledge about the DMU is still not complete. In fact, we still know little about functional attributes at the component level; that is, the functional unit level (see Section 3.4). Section 4.2.3 showed that the functional role, or roles, that a component plays in an assembly identify its FD. This information is necessary to meet the functional requirement of geometric model transformation for simulation purposes because an engineer is used to refer to a function or a group of functions in a synthetic manner using a ‘component name’ or a simple expression ‘qualifying a component’.

7.2 Inference rules as domain knowledge

This functional knowledge, however, is not beyond our reach. In fact, positive facts that emerge from the qualitative study explained in Chapter 6 open new opportunities for reasoning and provide seeds to express new knowledge.

To demonstrate this, we take the example of a component X that has only two FIs. One of them is a *planar support*, while the other is a *threaded link*, and they participate to a functional group LC which has *assembly joint with threaded link* as its FC (see Figure 5.5). We can then rationalize that such a component is indeed a *nut*, identifying its FD.

It is worth noticing that the newly obtained fact is the result of the application of an inference rule inspired by the domain knowledge that states that ‘a statically valid component having only two functional interfaces: a planar support and a threaded link by means of an internal thread, and involved in a assembly joint with threaded link, is a nut’. This rule is not formalized in any RSs.

This type of reasoning can then propagate throughout the assembly model, building on newly acquired knowledge to generate new facts, until the knowledge base converges toward saturation. For instance, the application of another rule stating that ‘a statically valid component that links to a nut through a threaded link, and that has another functional interface which is a planar support, is a capscrew’, allows us to deduce that the component D in Figure 5.5 is indeed a capscrew.

The previously mentioned rules can be formally stated by means of First Order Logic (FOL) as follows:

$$\begin{aligned}
 \text{nut}(X) \Leftarrow & \text{component}(X) & \wedge \\
 & \text{threadedLink}(I_1) & \wedge \\
 & \text{planarSupport}(I_2) & \wedge \\
 & \text{boltedJoint}(LC) & \wedge \\
 & \text{innerForms}(X, I_1) & \wedge \\
 & \text{forms}(X, I_2) & \wedge \\
 & \forall I(\text{forms}(X, I) \implies I = I_1 \vee I = I_2) & \wedge \\
 & \text{regroups}(B, X) &
 \end{aligned} \tag{7.1}$$

$$\begin{aligned}
 \text{capscrew}(X) \Leftarrow & \text{component}(X) & \wedge \\
 & \text{nut}(Y) & \wedge \\
 & \text{threadedLink}(I_1) & \wedge \\
 & \text{planarSupport}(I_2) & \wedge \\
 & \text{outerForms}(X, I_1) & \wedge \\
 & \text{forms}(Y, I_1) & \wedge \\
 & \text{forms}(X, I_2) &
 \end{aligned} \tag{7.2}$$

In this formalization, each of *component*, *threadedLink*, *planarSupport*, *boltedJoint*, *nut*, and *capscrew* is a unary predicate, while *forms*, *innerForms*, *outerForms*, and *regroups* are binary ones.

We note that the variable I in Formula 7.1 is *bound*. Unlike the other *free* variables of the formula that appear in more than one condition of the

implication, linking them together, I appears only in one condition. It is thus universally qualified.

Even though the second rule expressed by Formula 7.2 does not mention components participation to a assembly joint with threaded link in an explicit manner, this information is implied from the fact that X links to a nut through a threaded link.

Those rules reflect expertise about the domain of discourse, thus they are not part of any RS, as they are not hypotheses made in view of a given state of the product. However, the incorporation of such rules in the knowledge base is compulsory if functional information of the model is to be explored sufficiently to meet the objectives of this work (see Section 3.4).

7.3 Reasoning alternatives

A mechanism should thus be established to account for such domain knowledge. One may suggest the formalization of such rules into algorithms in a similar manner to RSs. This would mean the expression of each rule through a code path that assesses the satisfaction of its conditions, before applying its action. However, such an integration at the implementation level leads to static rules. Each time a new rule is to be introduced, the code must be amended, and resources must be recompiled.

While hypotheses made through RS are independent of the particular domain of application, as static validity and determinacy apply to all disciplines of mechanical engineering, inference rules are closely related to specific types of industries and industrial practices. In fact, a list of functional interactions, i.e., FIs and FCs, can be deemed exhaustive and complete with respect to a given set of conventional representations of components. A list of their possible combinations to produce FDs becomes more difficult to set up because new categories of components may appear. Anyhow, such lists cannot be ensured to include all FIs, FDs used in all mechanical industries at present time since conventional representations are not standardized at present (see Sections 1.6 and 3.2). To cope with this heterogeneous configuration, considering also that the proposed approach builds up on conventional representations whose consistency has not been analyzed, it is important to dynamically adjust inference rules, add new ones, or remove existing ones.

7.3.1 Dynamic formalization of domain specific rules

It is thus advantageous to enable the dynamic addition and modification of such rules, in order to tune our reasoning in accordance with specific needs of the particular domain of application, e.g., aeronautic or automotive industries. The implementation of inference rules as static, hard-coded execution paths is thus inadequate for such a requirement.

The ability to formalize rules in terms of FOL formulas such as Formulas 7.1 and 7.2 incites us to make use of this uniformity. If the sought system can take such formalization as an input, and then use it to produce new knowledge, rules can be adapted and augmented according to the particular domain where they apply.

7.3.2 Problem decidability

In fact, algorithms —referred to as deductive systems— exist to treat rules and facts expressed in FOL, and deduce new knowledge from them [61, 33, 91]. However, FOL is proven to be *undecidable* [51, 170]. This means that although FOL deduction can always find the answer in infinite time if the answer is positive (FOL deduction is *complete*), and that answers are valid when they exist (FOL deduction is *sound*), no algorithm is guaranteed to find the answer to any given question in an infinite time of execution.

This is in fact the price of high expressiveness of FOL. If a decidable deductive system is to be found, some logic constructs, thus some logic expressiveness, is to be sacrificed. Efforts have indeed been paid in this direction, to come up with fragments of FOL that are decidable. In this work we are particularly interested in a family of formal logic languages that is referred to as Description Logic (DL) [28].

In fact, DL is a family of decidable FOL fragments¹ that allow the fine tuning of reasoning algorithms complexity as a compromise on the logic expressive power. Moreover, DL provided the theoretical basis upon which OWL —the ontology language recommended by W3C— is built. Section 7.4 promotes the use of OWL ontologies as the knowledge base containers in this work.

Algorithms with controllable complexities are developed to allow efficient reasoning using DL. This led to a variety of reasoners that implement those algorithms, either for commercial use such as RACER [78], or with open source licencing agreements such as HermiT [155], Pellet [156] and FaCT++ [169]. Different reasoners treat different variants of DL. Because of its simple, yet well-defined formalism, interfacing protocols to communicate facts and queries to reasoners are also established, such as DIG [15].

From the previous analysis and the above-mentioned reasons, the use of DL to formulate inference rules suggests itself as a natural choice so that algorithm complexity can be mastered. This is indeed an important point to stay consistent with the geometric issues addressed in Chapter 5 and the qualitative reasoning process set up in Chapter 6 so that the overall DMU enrichment process can be mastered from an algorithmic complexity standpoint. Section 7.5 will develop in depth on issues related to the formal

¹Some DL variants go beyond FOL capabilities, providing operators that require higher order logic, such as transitive closure of roles or fixpoints [28].

reasoning employed in our work. Before that, Section 7.4 addresses a closely related topic, that is knowledge representation.

7.4 DMU knowledge representation

In order to enable the reuse of the acquired functional knowledge about a DMU at different stages of a PDP, it must be formalized and stored in a persistent manner. This brings forth the question of how knowledge should be represented.

Unlike most of other information systems, in an expert system the choice of method to achieve knowledge persistence is highly related to other choices about problem solving. This is because the issue of reasoning cannot be addressed in disregard of that of knowledge representation.

In fact, the way knowledge is represented decides how this knowledge is made available, in which way it is structured, and what elements it is made of. Those choices highly influence what kind of reasoning can be made on that knowledge, and what other information can be driven from. Therefore, both problems are coupled, and often addressed together in what is referred to as Knowledge Representation and Reasoning (KRR).

Section 2.6.2 showed that the literature intensively used ontologies to represent additional non-geometric knowledge about the DMU. Even though little has been done beyond the representation of facts in the analyzed works, ontologies, particularly with the powerful semantics of OWL, were shown to provide solid grounds not only for the representation of functional knowledge, but also for reasoning on it. Earlier works in the domain of CAD [127, 157] that concentrated on inference mechanisms was based on inference engine technologies that was not formalized as DL currently is. Consequently, there was no reference to the algorithmic complexity of these processes. Additionally, these approaches were connecting design parameters to geometric models of components in rather loose manner which was preventing them from referencing the precise and appropriate geometric areas of components and the integrity of the geometric model was not necessarily preserved, i.e., a solid may be transformed into an object that is no longer a solid. KBE approaches (see Section 2.6.3) appeared as evolutions of these early approaches linking geometry and artificial intelligence techniques. KBE concentrates on quantitative approaches to dimension components or sub-systems, i.e., it refers to the form – behavior dependency. Here, the knowledge representation is qualitative and can be expressed symbolically, which is well suited with the use of ontology-based approaches.

In a tight connection to the choice of reasoning formalism, that is DL, explained in Section 7.3, we opt in this work for representing acquired knowledge in terms of ontologies, and using OWL-DL language. This has the following advantages:

- The use of OWL ontologies, as a recommended standardized language [5], enables communication channels with other services through the well-established paradigms of the Semantic Web. Services may either be providers to which some tasks can be outsourced, e.g., reasoners, or consumers that make use of our expert system, e.g., FE pre-processors;
- OWL has a semantic advantage over its counterparts Resource Description Framework (RDF) and RDF-S which it actually extends with agreed-upon high-level semantics. This enables the formalization of fact and rules in a rather intuitive manner;
- OWL-DL offers a good trade-off between expressive OWL-Full with poor computational properties, and rather efficient, but quite restrictive OWL-Lite;
- The use of OWL-DL enables a seamless integration with DL reasoners, as DL is the formal logic on top of which the language is constructed.

As shown in Section 2.6.2, an ontology has two components. The terminological part, or TBox noted \mathcal{T} , and the assertional part, or ABox noted \mathcal{A} . Both the TBox and the ABox constitute the knowledge base $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$.

The TBox \mathcal{T} contains concept names and role names, and restrictions on them. While the ABox \mathcal{A} contains axioms, which are individual names and their instantiations.

7.4.1 Ontology definition through its concepts and roles

The proposed ontology is identified by its Uniform Resource Identifier (URI)², and defined by its TBox, as it contains the domain knowledge (see Section 2.6.2). ABoxes, containing the model knowledge, are then generated and stored apart, for each analyzed model. We recall that model knowledge is only interpretable in the context of domain knowledge, as Section 2.6.1 has shown. Therefore, before the model is treated, the ontology, in terms of its TBox, is loaded.

All introduced concept and role names have the namespace `romma`. In this text, and for the sake of readability, we refer to the concept name `romma:Component` as simply `:Component`, assuming that `romma` is the default namespace.

For the sake of clarity, we use the *CamelCase* notation in our naming conventions. Names starting with capital letters refer to concept names, while those starting with small letters indicate role names. Individual names are written in all caps.

²<http://pagesperso.g-scop.grenoble-inp.fr/~shahwana/romma>

The TBox defines functional taxonomies (see Section 4.2.6) through concept hierarchies. The FD taxonomy, which is partially shown in Figure 7.1, is defined by a concept hierarchy rooted at `:Component`. FDs are defined at the leaf level of the hierarchy, such as `:Capscrew`, `:Nut` and `:LockingNut`. While the root of FI taxonomy is the concept³ `:Interface`, FIs are again defined at the leaf level by concepts such as `:PlanarSupport`, `:ThreadedLink`, and `:SpineLink`.

FCs are defined as subclasses of the concept `:FunctionalCluster`. Examples are `:BoltedJoint` and `:RivetedJoint`.

The relationship between components (instances of `:Component`) and interfaces (instances of `:Interface`) is defined through the role `:forms`, that relates a component to an interface it *forms*. This role represents the binary relation \mathcal{R}_f defined in Section 4.2.5. The role `:links` is defined to be the inverse role of `:forms` (an interface *links* a component). This role represents the binary relation \mathcal{R}_l defined in Section 4.2.5. A restriction is added to the TBox to ensure that one interface links exactly two components.

The role `:forms` is specialized into two sub-roles, namely `:innerForms` and `:outerForms`. This is to faithfully represent geometric configurations where the interface is not symmetric. Examples of asymmetric interfaces are threaded links, spline links, snug fits, etc. In this case, we distinguish the outer component that *outer-forms* the interface, from the inner component that *inner-forms* it.

The relationship between components and their functional groups (instances of `:FunctionalCluster`) is defined through the role `:regroups` where a functional group *regroups* a component. The role `:participatesTo` is defined as the inverse role of `:regroups`, i.e., a component *participates to* a functional group.

The ontology editor Protégé [3] is used in the context of this work to design the ontology. This framework allows the intuitive authoring of concepts, roles, and their hierarchies. Moreover, the seamless integration that Protégé provides with reasoners (embedded support for HermiT and FaCT++ version 4) enables an easy check of concepts satisfiability. This permitted the reporting of inconsistencies as soon as they appeared.

7.4.2 Ontology population with model knowledge

Once the qualitative reasoning demonstrated in Chapter 6 is done, the resulting functionally enriched CIG is translated into an ABox of the above-mentioned ontology. To this end, the ontology is first loaded. Then the following steps take place:

- All nodes of the CIG are defined as individuals. For each node, an

³For the sake of conciseness, the terms *concept* and *role* are used instead of *concept name* and *role name*, whenever this use is not ambiguous.

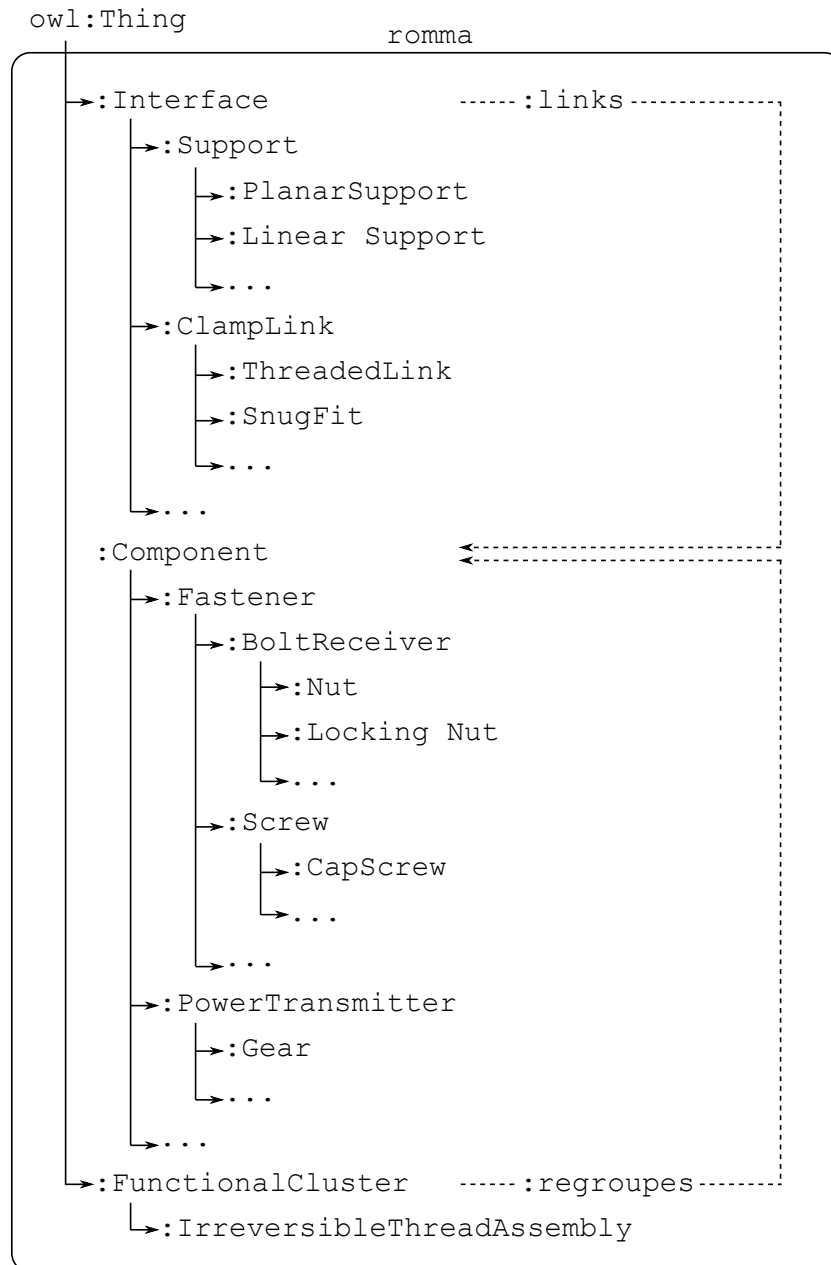


Figure 7.1: Partial graphical representation of **romma** ontology, showing concept and role names. Solid stroke arrows represent the concept hierarchy relationship, while dashed stroke arrows represent named object roles. The rounded rectangle delimits the name space.

axiom is added stating that the corresponding individual is an instance of `:Component`;

- All edges of the CIG are defined as individuals. For each edge, an axiom is added stating that the corresponding individual is an instance of the concept representing the FI associated to the underlying edge, i.e., a sub-concept of `:Interface`. Please note that as a result of the qualitative reasoning, each CI, thus each node of the CIG, is associated to one and only one FI. Indeed, all edges of CIG become instances of FIs defined as leaves in the taxonomy of interfaces;
- All recognized functional groups are defined as individuals. For each functional group, an axiom is added stating that the corresponding individual is an instance of the concept representing the group FC, i.e., a sub-concept of `:FunctionalCluster`. An assembly joint with threaded link (see Section 6.6) defined over a subset of the CIG nodes is an example of sub-concept of `:FunctionalCluster`;
- For each edge of the CIG, two axioms are added stating that the individual representing the interface links two other individuals representing the nodes at each extremity of the edge through the role `:links` or one of its sub-roles;
- For each recognized functional group, as many axioms as its participating components are created. Each stating that the individual representing the recognized functional group relates to the individual representing the corresponding component through the role `:regroups`. An example is the set of components belonging to an assembly joint with threaded link.

Now that the acquired knowledge is modeled using an ontology, the knowledge base is ready to be reasoned upon, generating new facts until saturation, i.e., until no new facts can be derived. The following section details this issue.

7.5 Formal reasoning to complete functional knowledge

Section 7.3 showed the merit of using a formal language to represent domain-specific expert rules. The choice of DL proves ideal for the needs of DMU functional knowledge. However, DL is a family of languages that vary with respect to their expressiveness, thus, with respect to their computational properties [17]. Variants of DL and the linguistic constructs they provide are visited in Appendix D. In this work, we employ the DL language *SHOIQ*

which is supported by OWL-DL, starting version 1.1, in agreement with our knowledge representation choice (see Section 7.4).

Even though, starting version 1.1, OWL-DL actually supports *SR_QIQ*-(*D*) [88], that is *SH_QIQ* augmented with complex restrictions on roles and data types constructs. In our work, we do not use these additional features. Disallowing data types in logical constructs aligns with the purely qualitative method advocated in Section 4.1 and described in Chapter 6. Moreover, avoiding the use of such costly constructs and restrictions spares the reasoning process a remarkable computational overhead [116].

DL family is backed by strong semantics that defines how different constructs of a given variant should be interpreted. Appendix D gives more details about the semantics of DL.

7.5.1 Inference rules in DL

Statements such as those expressed through Formulas 7.1 and 7.2 can be expressed in DL using General Concept Inclusion (GCI). A GCI in this sense defines an inference rule that applies across the underlying domain. Those rules are actually used to identify components FDs.

For example, the aforementioned formulas describing a nut and a cap-screw are translated into DL as follows in their respective order:

$$\begin{aligned}
 \text{Nut} \sqsubseteq & \text{Component} & \sqcap \\
 & \exists \text{innerForms. ThreadedLink} & \sqcap \\
 & \exists \text{forms. PlanarSupport} & \sqcap \\
 & \exists \text{participatesTo. BoltedJoint} & \sqcap \\
 & = 2 \text{forms} &
 \end{aligned} \tag{7.3}$$

$$\begin{aligned}
 \text{Capscrew} \sqsubseteq & \text{Component} & \sqcap \\
 & \exists \text{outerForms. (ThreadedLink} \sqcap \exists \text{links. Nut)} & \sqcap \\
 & \exists \text{forms. PlanarSupport} &
 \end{aligned} \tag{7.4}$$

In connection to what has been argued in Section 4.2.6, the hierarchical structure of the FD taxonomy allows inferences to gradually identify a component functional class. A rule can either describe an FD in one statement, or express an intermediate concept only, that leads to the definition of an FD when another rule is applied. Those intermediate concepts can be either inner node (nodes at upper level than leaves) in the FD taxonomy, or auxiliary concepts introduced to the ontology to allow the reuse of inference rules, independently of the FD taxonomy. In the latter case, the intermediate concepts holds no intrinsic functional meaning.

For example, the concept name `:BoltReceiver` can be used to refer to female threaded fasteners. It can then be described as follows:

$$\begin{aligned}
\text{BoltReceiver} \sqsubseteq & \text{Component} & \square \\
& \exists \text{innerForms.ThreadedLink} & \square \\
& \exists \text{forms.PlanarSupport} & \square \\
& \exists \text{participatesTo.BoltedJoint} &
\end{aligned} \tag{7.5}$$

The previous description includes any component with a threaded hole that receives another threaded shaft. The description of a nut can then be narrowed down as follows:

$$\begin{aligned}
\text{Nut} \sqsubseteq & \text{BoltReceiver} & \square \\
& =2\text{forms} &
\end{aligned} \tag{7.6}$$

The above-mentioned approach allows us to reuse the concept described in Formula 7.5 to describe other concepts, without the need of re-writing the whole statement each time the concept is reused. For example, a `:InnerNut` can be described as a `:BoltReceiver` with further restrictions:

$$\begin{aligned}
\text{InnerNut} \sqsubseteq & \text{BoltReceiver} & \square \\
& =2\text{forms.PlanarSupport} & \square \\
& =3\text{forms} &
\end{aligned} \tag{7.7}$$

This states that a statically valid component (as defined in Chapter 6) that has only three interfaces: a threaded link, two planar supports, and that takes part to an assembly joint with threaded link, is an inner nut (see Figure 7.2). A locking nut can then be defined as a nut, whose neighboring component is an inner nut.

$$\begin{aligned}
\text{LockingNut} \sqsubseteq & \text{Nut} & \square \\
& \exists \text{forms}(\text{PlanarSupport} \sqcap \exists \text{links.InnerNut}) &
\end{aligned} \tag{7.8}$$

We note that individual rules are not definitions. This means that each rule is actually an implication rather than an equivalence. This allows us to describe a given FD by means of two or more rules. If an individual satisfies any of those rules, it is identified as belonging to the corresponding FD. An example is to add another GCI stating that *a statically valid component that forms a threaded link by means of an external thread, and that has at least one planar or conic support, and that participates to a bolted joint, is a capscrew*, as follows:

$$\begin{aligned}
\text{Capscrew} \sqsubseteq & \text{Component} & \square \\
& \exists \text{outerForms.ThreadedLink} & \square \\
& \exists \text{forms}(\text{PlanarSupport} \sqcup \text{ConicSupport}) & \square \\
& \exists \text{participatesTo.BoltedJoint} &
\end{aligned} \tag{7.9}$$

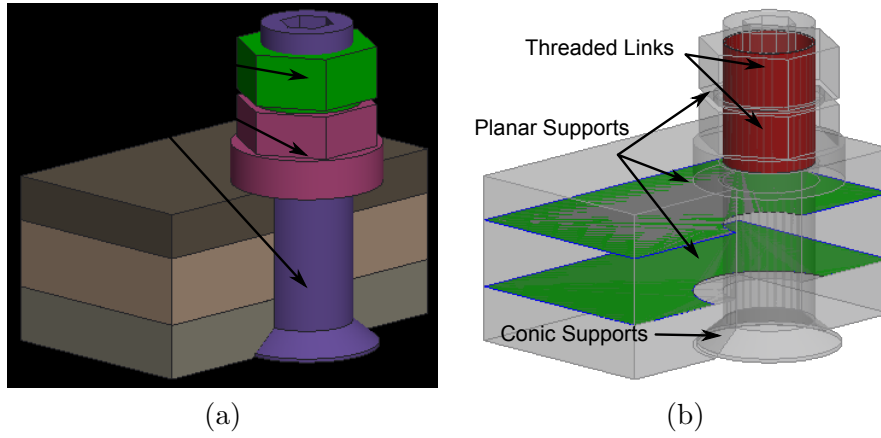


Figure 7.2: (a) A bolted joint (which is an assembly joint with threaded link), involving a capscrew, a nut, and a locking nut. According to inference rules, the nut would be identified as an inner nut. (b) Some of the FIs involved in the assembly.

By adding such a rule, a capscrew can still be identified, even when it is not connected with a nut. While the rule stated by Formula 7.9 is still valid.

It is worth noticing that all of the above-mentioned rules are expressed using only *SHOIQ* constructs. They are therefore expressible in OWL-DL. Again, we use Protégé to define expert rules, checking ontologies consistency and the satisfiability of concepts at each stage.

During a testing phase, Protégé is also used as a workbench, creating example ABoxes and using its reasoning capability to assess results completeness. The real feeding of the ABox axioms of a given model, however, is done from within our application, and not through Protégé. This will be developed in more details in Section 7.5.4.

In spite of the soundness of inference rules, expected results cannot be obtained by the mere instantiation of individuals and their relations as shown in Section 7.4.2. This is because of two inherent reasoning characteristics of OWL and DL that are the Unique Name Assumption (UNA) and the Open World Assumption (OWA). The upcoming two sections develop on these issues.

7.5.2 The unique name assumption

The Unique Name Assumption (UNA) assumes that distinct terms denote distinct individuals [135]. OWL, however, does not make this assumption. In fact, a URI is not required to be unique for a given entity in the Semantic Web philosophy. OWL even has the `owl:sameAs` property to denote

individuals identity, even though having two different URI, i.e., names.

Most of DL reasoners thus do not account for this assumption by default. This prevents making simple judgments, such as those on lower bound cardinality restrictions. For example, even if an individual x is declared to relate to individuals y and z through the role r in the knowledge base ABox, the conclusion that x is an instance of $\geq 2r$ cannot be made, in the absence of further knowledge. This is because names x and y may actually refer to the same individual.

To work around this issue, the UNA should be made explicitly. This can be done by means of *SHOIQ* constructs by defining a singleton for each individual, i.e., a concept having only one individual, then declaring those concepts to be mutually disjoint. This is made possible through the nominal construct of *SHOIQ* that allows rules to describe a concept by listing its individuals.

OWL defines the property `owl:differentFrom` that declares two names as referring to two distinct individuals. It also offers the `owl:AllDifferent` construct that declares a mutually distinct individuals. These OWL constructs are no more than syntactic shorthands to creating mutually disjoint singletons. In this work, however, the UNA is made using *SHOIQ* primitive constructs to guarantee compatibility with the communication protocol used (see Section 7.5.4).

7.5.3 The open world assumption

The Open World Assumption (OWA) assumes that facts which cannot be proven to be true remain unknown. This is in contrast to the Closed World Assumption (CWA), where facts that cannot be proven to be true are assumed false.

Under the OWA, partial knowledge is permitted, and conclusions made at some point cannot be falsified by the introduction of new facts to the knowledge base. While under the CWA, knowledge is assumed complete at each point of the reasoning. This leads to invalidate some conclusions that are made in absence of certain facts, once those facts are introduced to the knowledge base [75].

The OWA thus has the advantage of allowing partial knowledge, while keeping temporal consistency of the knowledge base. The CWA requires complete knowledge, that might not be available in the context of the Semantic Web services and applications. For this reason, OWL and DL make the OWA.

This assumption of an open world, however, may hinder reasoning. In fact, in the absence of some *closure* measures, inferences such as concept negation are not possible. The fact that an individual x , cannot be proven to belong to the concept C , does not mean that it belongs to its complement $\neg C$. In fact, x belongs to neither concepts. The same applies to upper

bound cardinality restrictions, the fact that an individual x is stated to be involved only once in a role r does not prove its membership of the concept $\leq 1r$, unless another involvement in the same role shows to be impossible.

This problem, however, can be solved by means of some sort of local closure of the world of discourse. For example, if the individual x is proven to be an instance of the concept D , while D and C are stated to be disjoint in the TBox of the knowledge base $C \sqsubseteq \neg D$, then $\neg C(x)$ becomes provable (and we write $\mathcal{K} \models C(x)$). This is because x cannot belong to D and C at the same time.

To work around this issue in the proposed approach, we allow for local closure of the knowledge base at the TBox level by defining concepts of different taxonomies (FI, FD and FC) to be mutually disjoint at each level of the taxonomy. Moreover, the closure is ensured at the ABox level, and for each treated model individually, by explicitly stating how many interfaces a component forms, and how many components a function cluster regroups. For instance, a component $:C$ (see Figure 5.5) can be stated to have exactly two interfaces as follows $\{C\} \sqsubseteq =2\text{forms}\top$. Again, nominal definition of a singleton (namely $\{C\}$) is used here, as enabled by the nominal construct of *SHOIQ* (see Section 7.5.2).

7.5.4 Integration of DL reasoners into application framework

As mentioned earlier in Section 7.3.2, efficient and robust algorithms are established in order to reason upon DL knowledge bases. The choice of OWL-DL as ontology language allows us to use these algorithms to saturate the knowledge base with valid new facts.

DL reasoning algorithms are implemented in terms of *reasoners*, which are expert systems that receive the knowledge base as an input, before inference algorithms are run against this knowledge in order to answer user's queries.

To allow the communication of facts and rules, as well as queries, a reasoner provides the software developer with interfaces to client systems. We distinguish two types of communication channels:

Application level communication. This is done through a well-defined API, and by means of a software library offered by the reasoner;

Network level communication. This is done through a well-defined network protocol, and by means of server applications offered by the reasoner.

The first solution, on the one hand, is suited for standalone software applications that are characterized by transparent communications with a minimal overhead. However, and in spite of efforts for standardization [87], different reasoners still define different APIs. This is partially because of the

difference of implementation techniques, e.g., programming languages, used by each reasoner. This disadvantages makes it impractical to switch between reasoners once the binding is done, hence it becomes tedious to switch from one reasoner to another to be able to evaluate their performances.

The second solution, on the other hand, goes through standardized communications. A reasoner that provides a network interface implements a well-defined protocol to this end. This allows the software developer to use an arbitrary reasoner, with complete abstraction of the implementation technique at the server and the client sides, as long as both sides implement and use the same protocol. Moreover, a different reasoner can be used, even after the interface is built, given that it provides the developer with a network interface. Following the client-server paradigm, this solution enables the assignment of dedicated reasoning servers for industrial as well as research applications.

A protocol for DL reasoners network communication actually exists, DL Implementation Group (DIG) is the *de facto* DL reasoner network communication standard [15]. It is used by many of them such as RACER, Pellet, and FaCT++ to name only a few.

To be able to evaluate the above-mentioned reasoners, we choose to use the architecture solution of network communication channel, implementing the DIG interface from a client perspective, to insert formal reasoning processes in the proposed approach as shown in Figure 7.3.

To this end, and for a given execution of our application, a connection to the reasoning server is then made, and a new knowledge base is initialized. The ontology TBox, translated into DIG *tell*-commands, is read. A tell-command informs the reasoner about rules (in terms of GCI) and facts (in terms of axioms). The ontology tell-commands is then submitted to the reasoner. After the geometric and qualitative analysis demonstrated in Chapters 5 and 6, the ABox of the knowledge base is also submitted to the reasoner, again in terms of tell-commands. Once all relevant facts are declared to the reasoner, i.e., the functionally enriched CIG as interpreted in Section 7.4.2, the individual names are declared unique (see Section 7.5.2) and the world of discourse is locally closed (see Section 7.5.3) using the relevant rules communicated as tell-commands. The reasoner is then queried by means of *ask*-commands about instances belonging to different FD classes. An ask-command requests the reasoner about a specific piece of information. The reasoner returns, for each ask-command, a list of individual names that belong to the requested concept, that is an FD in our case. These lists are used to annotate corresponding components with their respective FDs. Figure 7.4 illustrates different stages of this process, showing communication messages between the proposed application and a dig server in terms of a sequence diagram.

The above-mentioned scheme is totally independent of the underlying reasoner implementation. To carry out our experiments, we used both

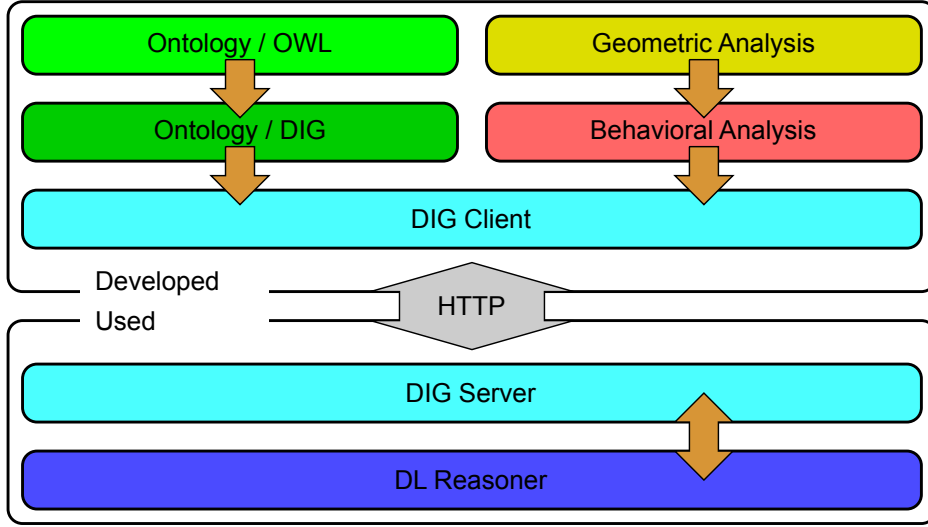


Figure 7.3: A diagram showing the architecture of the developed application, and its interface with the reasoning server. The diagram shows communication channels between different modules and applications.

FaCT++, a C++ implementation of a $\mathcal{SHOIQ}(D)$ reasoner [169], and Pellet, a Java implementation of a $\mathcal{SROIQ}(D)$ reasoner [156].

To allow the submission of the TBox to the reasoner, the ontology, expressed in OWL-DL, needs to be translated to DIG tell-commands first. Even though version 3 of Protégé provided an option to generate the DIG code corresponding to a given ontology, experiment showed that this code is broken. Protégé 3 actually communicates a different code when it binds to a DIG server, a property that was dropped in Protégé 4. For the purpose of this work, the TBox of the OWL ontology is translated manually to DIG tell-commands, while the rest of ABox tell-commands are formulated inside our application, in accordance with the facts acquired at the qualitative behavioral analysis stage. In order to complete the functional knowledge about the model on hand, the application queries the reasoner about instances belonging to concepts that represent FDs. A query is thus created per FD. Queries are formulated in terms of ask-commands. This functionality is implemented by the class `DigOntologyHandler` in our code.

For each query the application sends, the reasoning server sends back a set of individual names that belong to the FD in question. Individual names are then mapped to their corresponding components and components are annotated with the FD.

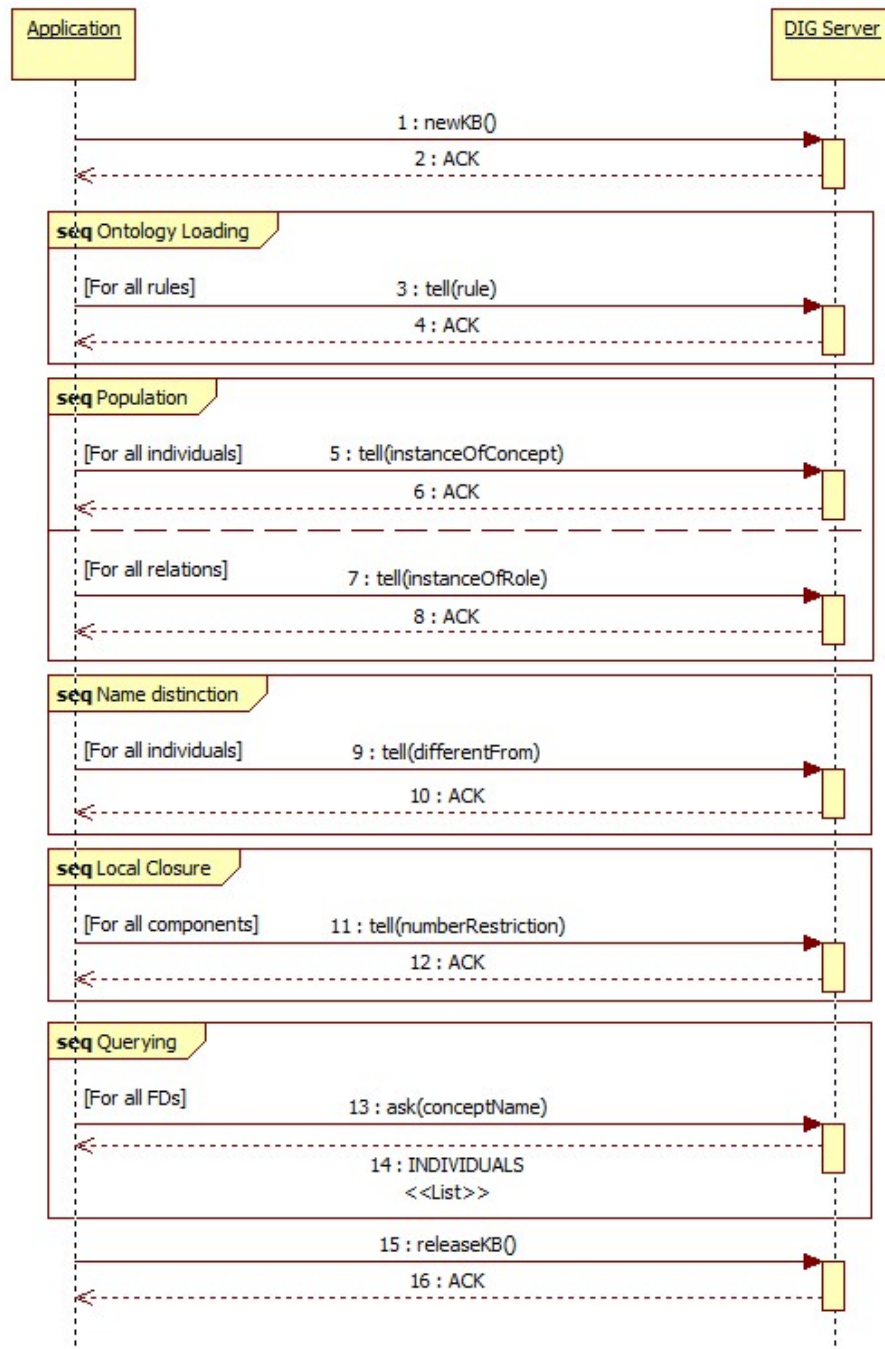


Figure 7.4: A sequence diagram showing communication between the proposed application and the DIG server.

7.6 Conclusions

After their identification at the functional interaction level, functional properties still need to be identified at the functional unit level. This can be materialized by assigning relevant FDs to their corresponding components. Section 7.1 emphasized this need to complete the functional enrichment process of the DMU.

Even though this information is still not available when the qualitative behavioral study, described in Chapter 6, concludes, Section 7.2 showed how this knowledge can be obtained. The assignment of FDs to their respective components showed to be the logical entailment of acquired knowledge at the interaction levels, i.e., the FI and FC levels, coupled with inference rules that are inspired from a particular domain of application. Indeed, while the qualitative analysis process refers to a behavior to filter out some FIs, it brings in a consistent set of dependencies between shapes, behavior and function at the level of interfaces between components. Section 7.2 showed how these dependencies at the component interface level could be extended to the component level through inferences.

Section 7.3 showed that a pre-defined set of such rules cannot be deemed complete for all mechanical engineering disciplines, thus the need for a dynamic adaptation of inference rules to the context in which a given DMU is defined. This requirement narrowed down the choice of the method to be used to define inference rules to formal logics that brings such agility. Although FOL suggested itself as a theoretically grounded solution, its unpredictable computational behavior made it an inconvenient choice. The advantage of expressing rules using DL, as a family of formal languages, is its well-understood computational properties. This is an important point to produce an application framework that can be mastered from an algorithmic complexity standpoint, both at the geometry and knowledge processing levels.

The choice of rules expression languages was made in close connection to the choice of knowledge representation shown in Section 7.4, and the choice of reasoning formalisms developed in Section 7.5. Our OWL-DL ontology was introduced in Section 7.4.1, before Section 7.4.2 showed how a given model is instantiated in view of its concepts and roles.

Section 7.5 demonstrated how inference rules can be formalized by means of DL *SHOIQ* constructs. Reasoning obstacles and their workarounds were then explained in Sections 7.5.2 and 7.5.3.

After establishing the reasoning process theoretically, Section 7.5.4 provided an insight into some implementation issues, such as the use of third party DL reasoners, and how knowledge, queries, and answers were communicated back and forth to such reasoners.

This chapter concludes the presentation of our purely qualitative approach. As a result of such a process, the DMU model is now restructured

geometrically to allow algorithms to recognize functional interaction zones (see Chapter 5) and identify some components using their FD. After interactions were recognized geometrically, their behavior was analyzed to recognize their functional attributes. Interaction zones, identified by CIs, and interaction groups, identified by components sets, were annotated with their functional semantics, i.e., their respective FIs and FCs (see Chapter 6). As a final stage to a complete functional interpretation of the DMU, components as functional units were annotated with their applicable FD, as part of a formal reasoning process. Indeed, this overall process describes the adequate data structure of a *functional component* as mentioned at Section 7.5.1

Chapter 8

Results and Comparative Study

In this chapter we evaluate the proposed application of DMU enrichment with functional information through different use cases, ranging from simple illustrative examples up to industrial scale models.

The general application architecture is first visited in Section 8.1. Section 8.2 walks through and comments the result of a range of examples. A successful case study of the integration with a template-based simulation pre-processor is demonstrated in Section 8.3.

8.1 Application architecture

Figure 8.1 shows the major modules of the proposed application. It also shows modules that the application uses to deliver its output. The system comprises three main modules that are the geometric processor seen in Chapter 5, the qualitative analyzer seen in Chapter 6 and the semantic annotator seen in Chapter 7. The three modules communicate by means of the CIG that is enriched with different levels of information according to the processing stage.

Open Cascade Technology [149] is used in the geometric processor to read a STEP file and perform primitive geometric and topological operations where the CIs between components are identified and the CIG is then generated. The geometric processor is used again in the semantic annotator to write down the STEP file, now enriched with the CIs between components and the functional knowledge about components. At this stage, the geometric model of each component is structured with all its CIs attached.

As shown in Section 7.5.4, a DIG reasoner is used, such as FaCT++ [169] or Pellet [156] to apply rules of the ontology defined in Section 7.4.1 onto the functional knowledge obtained through the qualitative analyzer, generating

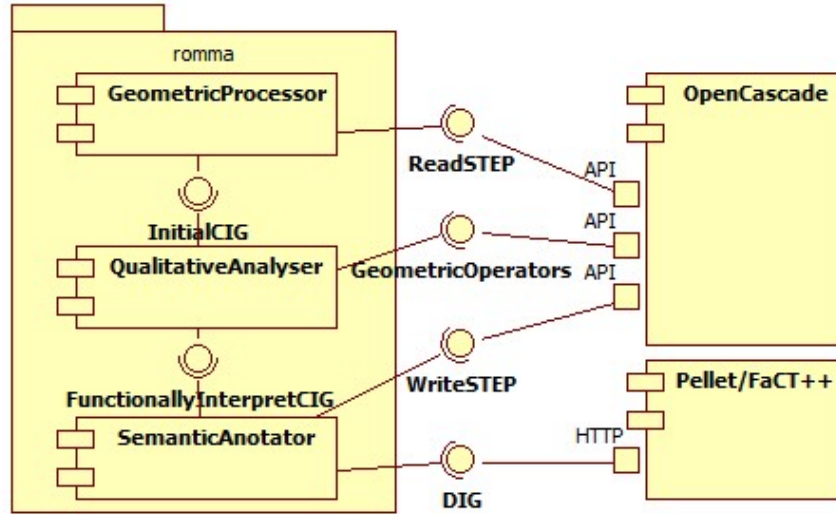


Figure 8.1: Component diagram showing the developed solution as encapsulated in package `romma`, along with external libraries that take part in the proposed approach.

new facts that relate components with their FDs.

Other software libraries have also been used in the developed application, such as Xerces-C++ [6] to parse and generate Extensible Markup Language (XML) strings, and cURL [2] to handle network communications.

The system takes as an input a DMU as represented by its geometry in a STEP format. In fact, most commercial CAD applications provide an interface to export CAD models to this format, as it is an ISO standard. Even though other information rather than geometry can be encapsulated in a STEP file, we only consider the geometry and topology of each component of the DMU. They represent closed B-Rep surfaces forming a volume each that represents a component.

The final output of the proposed system is again a STEP file. However, the generated STEP file differs from the read one in two aspects:

- The new geometry of the DMU is restructured according to functional interaction zones between components, and with respect to the functional breakdown seen in Section 4.2.3. That is, contacts and interferences are imprinted onto the original surfaces of each component participating to the interaction, generating new curves, thus new faces in the geometric model of each component;
- The new STEP file is semantically annotated. This is done in a tight

relation to the ontology defined in Section 7.4.1. Functional interaction zones are now named after their FIs designators as represented in the ontology. At this stage, one FI is associated with one CI; Moreover, functionally recognized components are also named according to their unambiguous FD, as borrowed from the same ontology. The connection to an agreed-upon ontology guarantees the meaningfulness of this supplementary information in the outcome DMU.

The proposed software can either be used as a command line application, or as a software library. In both cases, it will need a running DL reasoner server, supporting the DIG interface, in order to run properly. Documentation of the software API is available online¹.

8.2 Application to industrial examples

The proposed application has been run against different examples to evaluate its robustness. Tests included primitive DMUs that did not necessarily convey an industrial meaning, as well as full-scale industrial examples. In the first case, the system validity to generate coherent results with respect to the objectives set in Section 3.4 has been evaluated. In the second case, application scalability has been put to test using industrial use cases.

At first, we consider an illustrative example of a simple DMU. It is described purely geometrically and can be interpreted as a bolted joint assembly. The assembly consists of three plates fastened together by means of a capscrew, a nut, and a locking nut. We recall that those denominations are not available at the outset of the DMU analysis. A cross-section in the assembly model is shown in Figure 8.2. This figure also shows the CIG corresponding to the studied DMU, and generated by the geometric analyzer. RS I analysis is reflected in the elimination of statically invalid interpretations, namely both *spline links* FIs in this case. RS II comes then to filter out FIs leading to unnecessary static indeterminacy. This leads to the elimination of *snug fits* and the *self-locking fit*. Finally, RS III identifies an internal load cycle. This leads to the labeling of the group of components that participate to this cycle as an *assembly joint with threaded link*.

Once functionality is determined at the interaction level, the CIG is passed to the semantic annotator, which loads the ontology and connects to the reasoning server. The CIG is then translated into facts, in light of the ontology concepts and roles as shown in Section 7.4.2. After the reasoner is fed with available knowledge, it is inquired about FDs of components. Figure 8.3 shows recognized FDs as a result of the reasoning process. In addition to the *capscrew* in green and the *nut* in blue, the application also recognized the *locking nut*, colored in red. In fact, even though both the nut

¹<http://pagesperso.g-scop.grenoble-inp.fr/~shahwana/StepByStep/>

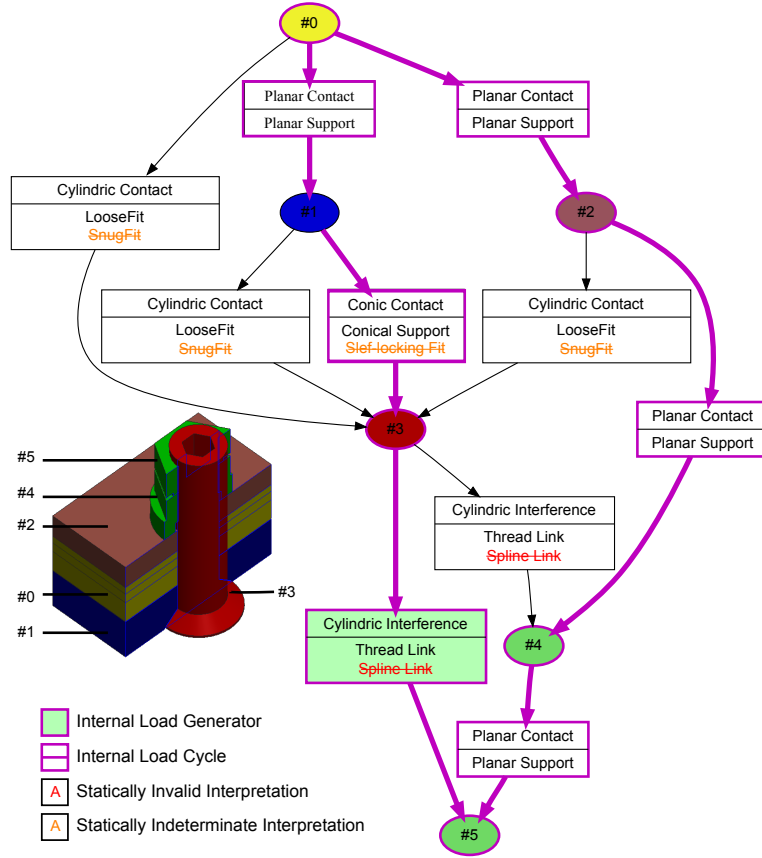


Figure 8.2: A cross-section in a simple bolted joint example tying up three plates, along with its corresponding CIG as generated by the geometric analyzer. Functional interpretations are also reduced to one FI per CI, and an internal load cycle is recognized as a result of RS qualitative analysis.

and the locking nut have the same shape, they are distinguished based on their CIs and FIs. The *locking nut* has two CIs whereas the *nut* has three. It has to be noticed that in a standard setting of a bolted joint with a single nut, this nut has only two CIs. It is because this nut is in contact with another nut that it functionally becomes a *locking nut* whereas the nut it is in contact with is functionally designated as a *nut* even though it has three CIs. To enable this distinction, the auxiliary concept of a *general nut* needs to be introduced to the ontology. A locking nut is then defined as a general nut that forms exactly one planar support with another general nut.

This simple example shows how FDs are influenced by FIs as well as other neighboring components: a complexity that can be handled with the reasoning mechanisms of the qualitative analysis and the inference mecha-

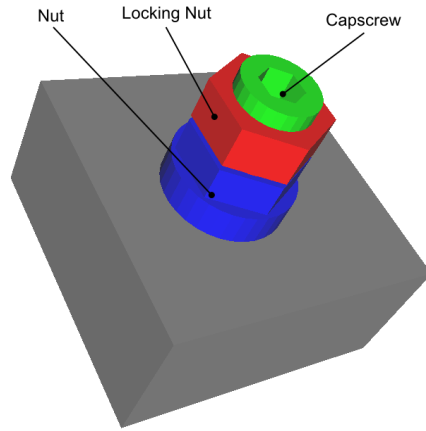


Figure 8.3: A bolted joint assembly showing a recognized *capscrew* in green, a recognized *nut* in blue, and a recognized *locking nut* in red.

nisms associated with the proposed ontology.

The model of the centrifugal pump, first introduced in Figure 1.5, provides a more elaborate example. The DMU contains 43 components. The geometric analysis of this DMU thus generates a CIG with 43 nodes. Model components, as represented by CIG nodes, are connected through 100 edges, that is 100 CIs. The CIs identified are of types surface contacts (planar and cylindrical), cylindrical interferences, linear contact.

Figure 8.4 shows the result of the running of the proposed application against the centrifugal pump model. This figure shows how the following FD could be recognized: a *capscrew* in green, *nuts* in blue, *studs* in yellow, *plug screws* in cyan and a *set screw* in magenta.

We note that since the capscrew, the studs, and the set screw all have an outer thread that participates to a threaded link, they are classified as *threaded shafts*. However, the distinction between one FD and another is made in light of components participation to other FIs. A stud for instance is guaranteed to form only threaded links as FIs. The diversity of FDs processed in this example demonstrate the interest of ontology structure and its associated inferences that can be enriched easily to adapt to new categories of components. This is an efficient complement to the qualitative analysis module that is generic and builds upon basic mechanical concepts.

The example illustrated in Figure 8.3 showed that even if two components share exactly the same shape, they still can be interpreted differently. It is also worth mentioning that the form of the stand-alone component does not affect the judgment of its FD. In fact, what does matter is components interactions, reflected first at the geometric level by their CIs, and then interpreted functionally by means of FIs. For example, a nut is recognized

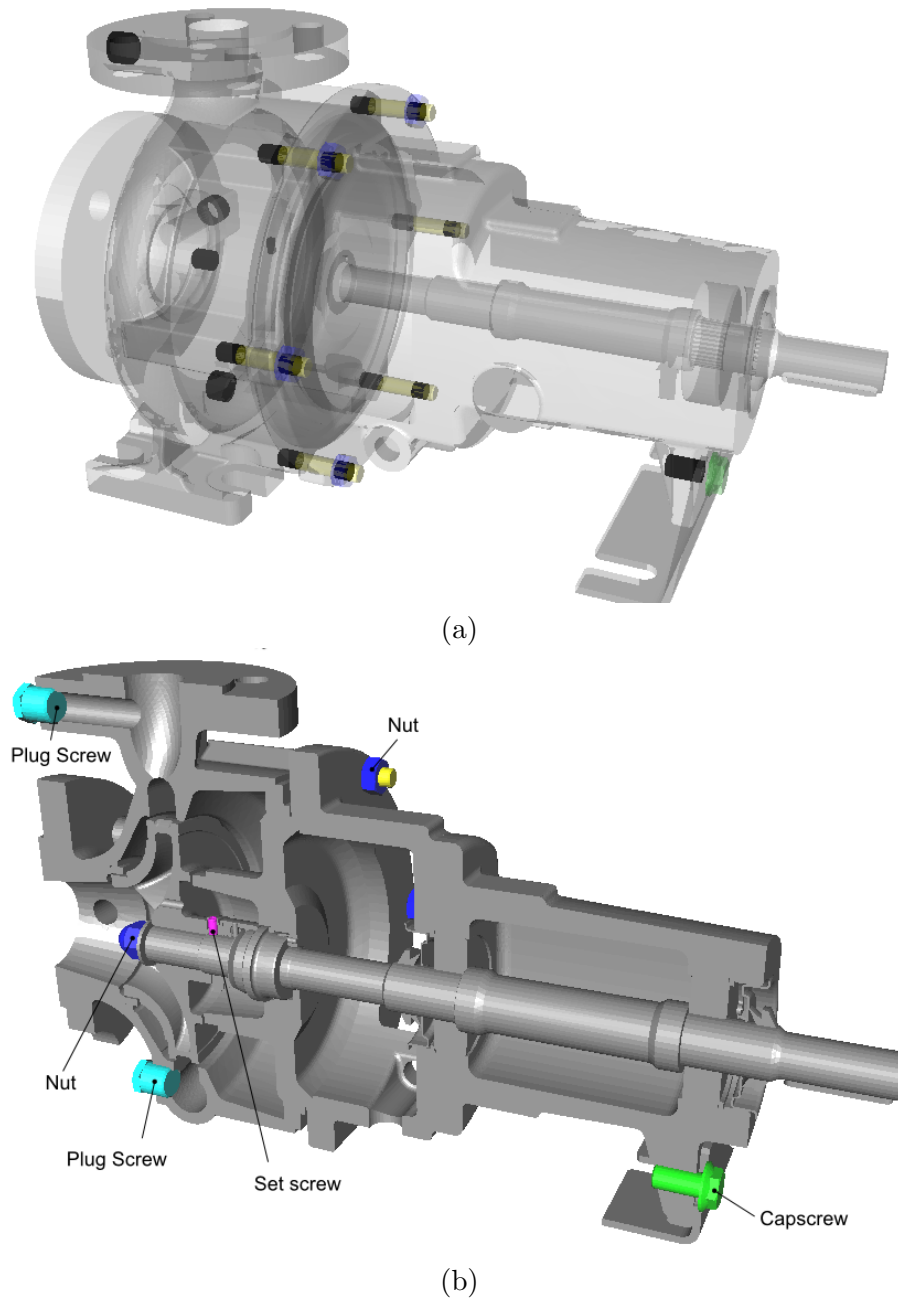


Figure 8.4: The example of the centrifugal pump after it has been treated by the proposed qualitative approach to detect its FD. (a) A semi-transparent rendering of the DMU, detected CIs of type interference are shown in dark black. (b) A cross-section cut has been applied to the generated DMU to show internal parts. Recognized components are a *capscrew* in green, *nuts* in blue, *studs* in yellow, *plug screws* in cyan and a *set screw* in magenta.

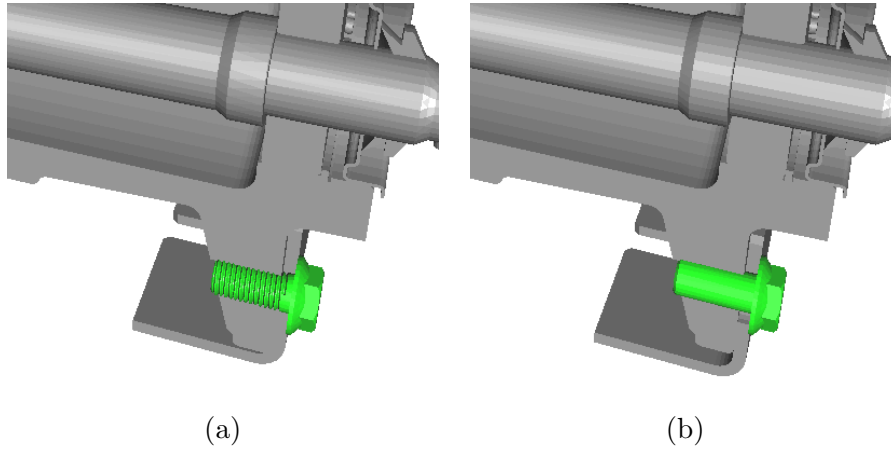


Figure 8.5: Two different conventional representations of an inner thread corresponding to a screw thread. (a) The inner thread is represented with detailed helical shape on the screw, while the outer thread is simplified as a simple cylinder on the housing. This leads to a complex interference zone, which highly depends on components relative position. (b) The inner thread, as well as the outer thread, are represented as cylinders, leading to a cylindric interference between these two components.

regardless whether it is a cap nut or a simple nut with hole, as long as it satisfies the nut functionality, as Figure 8.4 shows.

Figure 8.5a shows a different representation of the capscrew of the pump model, as compared to the one used in the first example, and illustrated closely in Figure 8.5b. In fact, different representations are due to different conventions, as discussed in Section 3.2. Our algorithm shows to be general enough to recognize both conventions, and interpret them correctly, as the figure shows. In fact, even though the convention illustrated in Figure 8.5a represents the thread in more details with a helical shape at the capscrew side, generating a fairly complex geometric interface, the geometric analyzer of the proposed application reduces it to a simple cylindrical interference to allow the subsequent qualitative reasoning to take place. The real shape of its geometric interface is highly dependent on positioning parameters that are usually relaxed during the designing phase, it has thus to be simplified before it is mapped to an interpretable CI. Here, the analysis based on the relative position of cylindrical surfaces is the key property of this simplification process.

Table 8.1 shows execution time for the example of the centrifugal pump, run on a machine with an Intel® Core™ 2 Duo processor at 2.40GHz and 4GB of memory. It also shows how execution time varies with respect to the number of detected FDs that decides the size of the underling ontology;

Table 8.1: Execution time of the proposed method (in seconds), run against the example of the centrifugal pump, as a function the number of detected FDs.

Number of detected FDs	3	4	5	6
Execution time (seconds)	15.69	16.26	16.34	16.54

i.e., the number of its rules.

Another example that is used to evaluate the proposed method scalability is the root joint example. This structure is a small subset of an aircraft structure connecting a wing to the aircraft fuselage. It is a use-case set during the ROMMA project [1] submitted by Airbus as project partner. Figure 8.6 shows the model of the root joint as an output of the suggested application. The DMU of the root joint contains 148 components. The geometric analysis of such a model shows that components connect to each other through 512 geometric interfaces.

It is worth noticing that the execution time of the geometric processing and qualitative analysis is negligible when compared to that of the semantic reasoning, done externally to our application by means of a DL reasoner. In fact, even though DL reasoning complexities have well-known and understood bounds, those bounds are shown to be ExpTime-complete in the general case [27]. Many factors influence the DL reasoning time, among which the amount of provided facts, that is in our case a function of the number of components and CIs. Another important factor is the size of the ontology, which dictates the number of rules taken into account when the reasoning takes place (see Table 8.1). To alleviate the time complexity problem while dealing with large-scale industrial models, an ontology can be simplified to only include rules that define FD that are relevant to the studied model and phenomenon. Such an adaptive ontology has been applied to the root joint example to allow reasoning in a timely manner (less than one minute on a personal computer) while still giving relevant and accurate results. In the above-mentioned example, the ontology rules were reduced to only recognize capscrews, nuts, and locking nuts.

It has to be noticed however that the general ontology used efficiently against the centrifugal example, is also used against a sub-assembly of the root joint as shown in Figure 8.3, while keeping execution time reasonable (few seconds). In this case, and instead of reducing the number of rules in an ontology, the number of facts is decreased by examining only a sub-assembly of the whole DMU. Similarity and symmetry properties can then be used to generalize obtained conclusions to the rest of the DMU while keeping timely execution. Figure 8.7 shows execution time of the proposed method

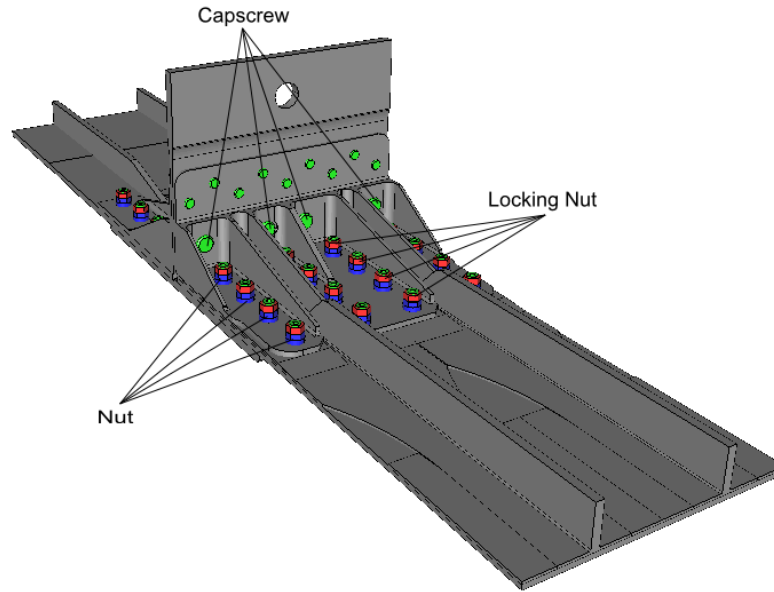


Figure 8.6: An industrial example of a root joint after it has been processed by the proposed application. Recognized FD in this model are *capscrews*, *nuts*, and *locking nuts*.

run against the example of root joint, as well as other sub-assemblies of the same structure. Execution time is plotted as a function of the number of components in each substructure. Experiments are run on a machine with an Intel® Core™ 2 Duo processor at 2.40GHz and 4GB of memory.

8.3 Integration with FEA pre-processors

The proposed approach has proved to integrate seamlessly with automatic FEA pre-processing task [42] as to meet its target. In fact, the output of the proposed application can be fed to a FEA pre-processor as its input. A pre-processor that is aware of the ontology we put forward in Section 7.4.1 can then read the produced functionally enriched DMU, in its STEP format, as well as the FD of each component, in order to prepare a simulation model, while taking into account the simulation objectives (see Figure 1.19). Functional information available in the produced model allows the pre-processor to robustly relate geometry at different levels, i.e., geometric interface, component, and component group, to functionality as defined in the ontology. The pre-processor can then choose the adequate simplifications and/or idealizations to apply on that particular geometric zone in lights of simulation

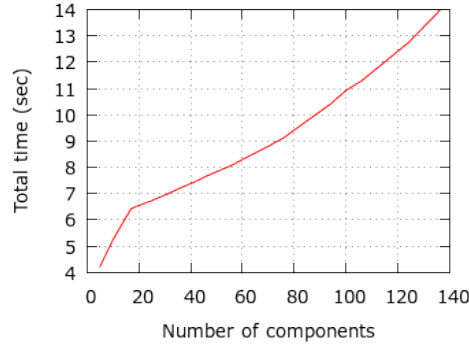


Figure 8.7: Execution time of the proposed approach against the example of root joint and its sub-assemblies, as a function of number of components.

objectives and hypotheses.

Figure 8.8 shows how the template-based approach proposed in [42] builds on functional annotations provided by the hereby proposed application to simplify a bolted joint connection in accordance with simulation purposes. To allow geometric pre-processing, a set of templates are defined, to which a set of transformations can be associated, according to simulation objectives. For example, the functional group of bolted joint is first recognized as it is labeled as a *assembly joint with threaded link* by the hereby proposed approach. It is thus matched to a template T. Links are also established between elements of T and functional group components and interfaces, based on their FDs and FIs. This is made possible because the DMU, restructured and functionally enriched through the proposed approach, is no longer a mere annotation of components but this annotation relates to the geometric structure of components using their FIs and load cycles that completely define each bolted joint. Once components and interfaces are matched to template elements, geometric transformations can be applied and adapted to the screw diameter, the number of plates tightened together, the type of screw head, the existence or not of locking nuts, i.e., the template is largely parameterized and becomes generic. Rather than selecting individually each component and performing low level geometric tasks on each component, the engineer can now select components based on their functional meaning, here the *assembly joint with threaded link*, which is very useful in the present case to transform the screws and nuts into connectors as needed for FEA simulation purposes.

In the illustrated case, and for the sake of structural simulation, the locking nut was first removed as its functionality was detected as secondary by the template T. The capscrew and the nut were then merged together, removing the binding threaded link. Loads are created as normal to the

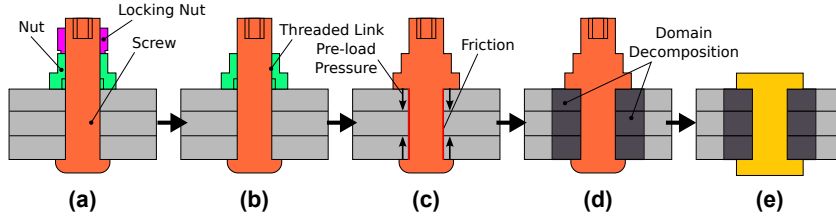


Figure 8.8: A template-based simplification of a bolted joint assembly for FEA simulation purposes [42]. (a) The original sub-assembly model annotated with FD and FI as an outcome of the hereby proposed application. Since it is recognized as an *assembly joint with threaded link*, the sub-assembly is matched with a template T. (b) The *locking nut* is removed, as recognized as a secondary FD in the context of *assembly joint with threaded link* by T. (c) The removal of the *threaded link* to merge the screw and the nut. (d) Domain decomposition takes place around the *cylindric support* interaction zone. (e) Screw head transformation extends the range of screws to flat-headed ones.

planar support, while friction areas are added under the screw head and the planar support of the nut. Then, the object resulting from the fusion of the capscrew and the nut is further simplified as a flat-headed fastener. Finally, the load cycle of each bolted junction gives access to the plates it tightens and a sub-domain can be created in these plates around the screw as needed to set frictions areas between the plates and adapt the FE mesh generation technique in that area. This generates a simulation model that complies with hypotheses and objectives, and readily allows the generation of the FEM.

Overall the proposed approach enabled the time reduction in processing this model from five days to interactively process this DMU as required by an engineer with the current software tools to one hour with the newly proposed approach using the template-based operator exploiting the enriched DMU with the functional information generated by the proposed approach.

8.4 Conclusions

In this chapter the proposed approach has been studied from a pragmatic standpoint. Developed algorithms and advocated methods have been put to test with concrete examples to evaluate their validity and scalability.

Results show the merit of a qualitative approach, which generates functional knowledge from a purely geometric model, based on an adaptive set of domain expert rules, while taking into account mainstream industrial practices and conventions. They also show potentials of enhancement and

optimization. The following chapter concludes this work, and presents some perspectives to extend this work.

Chapter 9

Conclusions and Perspectives

9.1 Conclusions

The proposed approach to structure and enrich DMUs with functional information has analyzed, in a first place the effective content of DMUs in terms of functionally related information available as well as other technological information that could be processed to meet our objective. Chapter 1 showed that B-Rep models of components where the generic basis available in any CAD or FEA software and other technological or functionally-related information was sparse and, generally inexistent. The DMU structure, i.e., its hierarchical description, as well as position constraints or kinematic connections between components are not intrinsic to a DMU. The DMU structure may not reflect its kinematic behavior and position constraints or kinematic connections between components, when available, are not intrinsically related to the interfaces between these components. Similarly, component names are not reliable information that can be related somehow to component functions. Consequently, the proposed approach has been based on the B-Rep models of components, positioned in 3D space independently of each other, as DMU representation that can be reliably used as input for our enrichment process.

The analysis of prior work (see Chapter 2) has shown that functional information has been processed mostly through top-down approaches, with function definitions loosely related to the geometric entities, i.e., faces, edges, vertices, of components. Strong dependencies, however, between shape – function – behavior is commonly recognized though not formalized in details from a geometric point of view. Feature-based approaches on standalone components have led to numerous applications, however, functional information has not been fruitfully addressed. Processing assembly models brought more information about their geometric interfaces but few contributions addressed this level and none of them focused on the enrichment with functional information. Design knowledge modeling and KBE

approaches added design and functional knowledge to components, essentially through interactive means, KBE approaches being more automatized but reduced to a narrow application range. Anyhow, the technological or functional information is loosely connected to the component shape and does not strongly influence the geometric structure of components. Where ontology-based approaches have been proposed, reasoning capabilities have rarely been proposed, KBE ones however extensively use dedicated behaviors for very specific applications. The proposed approach is bottom-up to take advantage of intrinsic and robust data as input, i.e., component shape and their relative positions. Also, it is tightly related to shape – function – behavior dependencies, which have not been precisely investigated to the best of our knowledge. Ontology-based reasoning processes bring a well formalized framework with algorithmic complexity characterization that brings more efficiency compared to the use of ruled-based systems used in the CAD area in prior work.

From these settings, Chapter 3 pointed out that frequently, a real component shape does not match its digital representation. This difference has to be taken into account since it influences the geometric interfaces between components, hence the reasoning processes that can be set up from digital models of components must take into account these differences. Because the concept of function tightly relates to the concept of interactions between components, the focus has been placed on the geometric interfaces between components. The differences between real and digital shapes has been formalized through the concepts of CIs and FIs and knowledge related to these interfaces has been structured through appropriate taxonomies (see Chapter 3 and Chapter 4). Indeed, the differences between real and digital interfaces between components end up as multiple interpretations that require a reasoning process to filter them out and generate a DMU enrichment process that can be automated.

The determination of the geometric interfaces between components is not a straightforward process, as pointed out in Chapter 5. This chapter has shown how the accurate definition of imprints of interfaces between components can be sped up and obtained. Though digital shapes of components lead to three categories of interfaces, i.e., contact, clearance and interference, their analysis has concluded that contacts and interferences are the only intrinsic categories that can be used initially to enrich an assembly model with functional information. This choice is consistent with respect to the definition of an approach that relies on intrinsic information. The taxonomy of FIs connected to the precise geometric description of interfaces over components is a first setting of the dependency between shape and function.

At this stage, components geometric models are structured as well as the assembly model through its CIG. All this information and the structured models can now be used to take advantage of dependencies between shape, function, and behavior to filter out the multiple interpretations existing at

some component interfaces. To this end, the concept of reference state has been introduced that can be associated with a wide range of DMU configuration throughout a PDP. Then, several qualitative behaviors have been described that rely on generic mechanical properties, i.e., static equilibrium of a component, load propagation cycles, and statically indeterminate configurations, and enable to filter out FIs (see Chapter 6). These reference states are automatically applied and stand for a first reasoning process performed algorithmically. Therefore, its efficiency can be analyzed and its termination can be established. These reference states, as well as the FIs cover only a subset of the possible interfaces between components. Consequently, the straightforward application of these qualitative behavioral models to an arbitrary DMU can lead to incomplete results if CIs are not falling into the proposed taxonomy. In a correct setting, however, the qualitative analysis produces a unique FI per CI, which strengthens the relationship between each CI and its corresponding FI. The DMU thus obtained is consistent at the interface-level and knowledge has been gained at the cluster-level, which structures further the assembly model.

Finally, the previous results are input in the second step of reasoning, which is based on a ontological approach. The previous taxonomies of FIs are associated with a pre-defined, generic taxonomy of FDs to lead to the assignment of an FD per component. This parameter becomes an identifier of a component that combines with its FIs and its geometry structured with the imprints of its CIs and other FIs of neighboring components as required by its FD to form the generic information characterizing functionally the component inside its DMU (see Chapter 7). The FD assignment process is obtained through a rule-based process. Inferences are expressed using descriptive logic whose algorithmic complexity is established. Consequently, the overall process of DMU enrichment is guaranteed to terminate and its algorithmic complexity can be mastered. The enrichment is now obtained at component-level as well as component cluster-level. This process is therefore well suited for industrial DMUs and particularly complex ones as they can appear in the aircraft industry. The enrichment thus obtained is robust, i.e., it is consistent and independent of user's interactions, since the enrichment process is automatic.

Chapter 8 has illustrated the results of the proposed approach with various examples. The DMU functional enrichment thus obtained has been successfully used in the context of FEA preparation processes. There, it can be demonstrated how the functional enrichment can contribute to the component selection process for fasteners and save a fair amount of time. Also, Chapter 7 and Chapter 6 have shown how they can be extended to analyze the consistency of a DMU.

9.2 Perspectives

Applications in Virtual Reality and Motion Planning

Although FEA model preparation is the first major objective behind this work, applications are not restricted to this and other applications in a PDP can benefit from the proposed approach. We show how our approach fits other applications such as virtual and augmented reality and robotic and motion planning.

VR Applications in PDP

DMU geometric model can be employed in VR applications, where the users are immersed in a virtual environment and they can manipulate the product and simulate its use and ergonomics. In return, virtual and augmented reality techniques, varying from simple visualization to fully-immersive environments, can be applied to PDP at different stages such as design and assembly/disassembly planning and simulations [38, 141, 150].

VR approaches use simplified and approximate physical models to model interactions between user avatars and other objects, and between objects themselves, to allow real-time simulation of such an interaction. These models are good enough to simulate a huge portion of expected interactions. However, and for certain cases, a particular physical model fails to provide realistic results. An example is contact simulation between rigid bodies where collision detection algorithms are used to recognize contacts, and generate appropriate forces. Such methods are usually based on objects tessellated model (see Section 1.5.2). However, for special cases such as shaft/bushing connections, simulation based on simple collision detection algorithms generates an unstable behavior of haptic devices, as simplified physical hypotheses and object dimensions are not compatible with the conventional representation of components in a PDP. In such a case, when the shaft approaches the housing, reciprocal forces are generated from the bushing edges that push the shaft away because the shaft and bushing diameters are either equal or closer to each other than the geometric deviation used in collision detection algorithms, thus preventing the desired sliding effect between these components.

Efforts have been paid to account for this inconvenience, where early work shows how to use objects and agent specific attributes in order to realize motion planning. Levison & Badler use behavioral object knowledge to build an *object-specific reasoner* to enable a high-level action planning [110]. Kallmann & Thalmann propose a virtual interactive environment in which *smart objects* define how they can be used and interacted with, by means of a set of possible states, conditions, and instructions [96]. This allow the adaption of the underlying physical model to the particular action to be performed, avoiding unrealistic effect such as the shaft/bushing phenomenon.

Jorissen & Lamotte generalize this approach to enable interaction between all objects and human avatars in a virtual environment [93].

All previously presented approaches require functional annotations of objects in order to assign them an appropriate behavior and interaction scheme at the correct location around each object. In the presented works this knowledge is provided during the design time of objects and virtual environments, which suffer problems as mentioned in Chapter 2 and Chapter 4.

To allow the application of VR and motion planning to large scale models, such as a product DMU, the manual functional annotation becomes cumbersome (as seen in Section 3.4). Our work proposes an automated method to boost functional annotations of objects and enable the location of interfaces between components as a complement of prior work [89]. Incorporating usage patterns based on objects functionality enables the application of the above-mentioned approaches to industrial scale models and environments because of the tight relationship between geometry, i.e., some local areas of components, function as assigned through the proposed approach, and behavior as needed for these simulations. Consequently, dedicated VR simulation models could be triggered whenever needed for an interaction during a VR simulation process. The use of appropriate VR component behavior become transparent for the user, it is interaction-driven.

Contributions to CAD-to-FEA transformations

In their recent work, Boussuge et al. [42] introduced a set of automated geometric transformations of CAD models in preparation for FEA. The transformations are based not only on the mere geometry of the model, but also on supplementary simulation-relevant annotations that go through functional groups of components down to the geometric zones that delimit functional interactions between pairs of components. This approach uses the work presented in this manuscript to structure the geometric model of components and connect this new structure to such functional annotations. The categories of components under focus were fasteners. This need to be extended with a larger range of categories of components to take advantage of the proposed principle. To this end, the qualitative approach needs to be extended with new reference states, particularly those that can refer to kinematics, i.e., relative movements between components. This would enable the identification of kinematic equivalence classes and their corresponding components in a DMU.

Reasoning with these additional reference states would open the possibility to set up new inferences to categorize components of type bearing, gears, couplings, etc.

Even though the scheme described in the manuscript joins KBE in some aspects, it is however a more robust approach. This is because functional

designations and functions are generic concepts in our approach. KBE aims at structuring engineering knowledge and processing it with symbolic representations [47, 143] using language based approaches. In this work, the focus is placed on a robust connection between geometric models and symbolic representations featuring functions, which has not been addressed in KBE and, therefore, could be used in KBE approaches to extend their range and improve their robustness when DMUs are modified during a PDP. Additionally, KBE approaches as well as CAD-to-FEA transformations show that the proposed approach can be regarded as a means to analyze and structure a DMU at various stages of a PDP.

Among these stages, the design process, and even the early stages, can benefit from the proposed approach to automate the enrichment of a DMU with functional information in a top-down manner. As pointed out in Chapter 2, many design approaches based on hierarchical decompositions face difficulties when refining the design downward to detailed geometric configurations. These hierarchical decompositions needs evolutions to meet the graph-based approach that is intrinsic to many mechanisms, as shown by the CIG, the load cycles, and the complexity of inference processes. The proposed DMU enrichment process is robust with respect to a range of conventional representations of components, the influence of these representations has not been addressed with respect to the enrichment process to evaluate how it can be further improved or how it is robust to variants of representations. Studying these issues would help setting up principles and/or standards for a more efficient processing of DMUs at many stages of a PDP. Similarly, studying the robustness of the enrichment process with respect to variants of representations of components open the possibility to design more tolerant software environments that would refer to a sketch-based paradigm though they would stay robust while being more user-friendly.

Appendix A

Fit Tolerancing and Dimensioning

In mechanical engineering, a *fit* refers to a mating of two mechanical components; one is a containing *housing*, referred to as the female part, and the other is a contained *shaft*, referred to as the male part.

In technical drawing, both shaft and housing have the same nominal diameter. Tolerancing, however, decides whether it is a snug fit, loose fit, or nondeterminate fit.

ISO defines how to annotate drawing with such information in a standardized manner. The tolerance is denoted by a tolerance code that constitutes of a letter followed by a number. The letter defines the deviation from the nominal dimension as a function of it. Small letters are used to dimension shafts by defining the maximal deviation $es = d_{max} - d_{nom}$, while when used for housings the letter is capitalized defining minimal deviation $EI = D_{nom} - D_{min}$. For example, the letter H means zero distance from the nominal diameter for the housing (referential housing), while the letter h means zero distance from the nominal diameter of the shaft (referential shaft).

Table A.1 shows tolerance letters for defining maximal and minimal deviation of shaft and housing dimension (respectively) as a function of nominal dimension.

Tolerance letters are followed by a number, defining tolerance quality. The number is proportional to the tolerance, thus disproportional to the manufacturing quality. Table A.2 shows 18 tolerance qualities defined by ISO as a function of the nominal dimension.

Since the machining the shaft is more precise, shaft tolerance is usually of higher quality rather than that of the housing. The combination of the tolerance codes of each of the shaft and the housing decides whether the fit is snug, loose, or nondeterminate.

Nominal dimensions are expressed in millimeters, after a symbol that

Table A.1: ISO deviation codes showing deviation of the nominal dimension in μm as a function of nominal dimension (in mm).

Lettre		c	cd	d	e	ef	f	fg	g	h
≤ 3		-60	-34	-20	-14	-10	-6	-4	-2	0
> 3	$\& \leq 6$	-70	-46	-30	-20	-14	-10	-6	-4	0
> 6	$\& \leq 10$	-80	-56	-40	-25	-18	-13	-8	-5	0
> 10	$\& \leq 14$	-95	—	-50	-32	—	-16	—	-6	0
> 14	$\& \leq 18$	-95	—	-50	-32	—	-16	—	-6	0
> 18	$\& \leq 24$	-110	—	-65	-40	—	-20	—	-7	0
> 24	$\& \leq 30$	-110	—	-65	-40	—	-20	—	-7	0
> 30	$\& \leq 40$	-120	—	-80	-50	—	-25	—	-9	0
> 40	$\& \leq 50$	-130	—	-80	-50	—	-25	—	-9	0
> 50	$\& \leq 65$	-140	—	-100	-60	—	-30	—	-10	0
> 65	$\& \leq 80$	-150	—	-100	-60	—	-30	—	-10	0
> 80	$\& \leq 100$	-170	—	-120	-72	—	-36	—	-12	0
> 100	$\& \leq 120$	-180	—	-120	-72	—	-36	—	-12	0
> 120	$\& \leq 140$	-200	—	-145	-85	—	-43	—	-14	0
> 140	$\& \leq 160$	-210	—	-145	-85	—	-43	—	-14	0
> 160	$\& \leq 180$	-230	—	-145	-85	—	-43	—	-14	0
> 180	$\& \leq 200$	-240	—	-170	-100	—	-50	—	-15	0
> 200	$\& \leq 225$	-260	—	-170	-100	—	-50	—	-15	0
> 225	$\& \leq 250$	-280	—	-170	-100	—	-50	—	-15	0
> 250	$\& \leq 280$	-300	—	-190	-110	—	-56	—	-17	0
> 280	$\& \leq 315$	-330	—	-190	-110	—	-56	—	-17	0
> 315	$\& \leq 355$	-360	—	-210	-125	—	-62	—	-18	0
> 355	$\& \leq 400$	-400	—	-210	-125	—	-62	—	-18	0
> 400	$\& \leq 450$	-440	—	-230	-135	—	-68	—	-20	0
> 450	$\& \leq 500$	-480	—	-230	-135	—	-68	—	-20	0
Lettre		C	CD	D	E	EF	F	FG	G	H
≤ 3		+60	+34	+20	+14	+10	+6	+4	+2	0
> 3	$\& \leq 6$	+70	+46	+30	+20	+14	+10	+6	+4	0
> 6	$\& \leq 10$	+80	+56	+40	+25	+18	+13	+8	+5	0
> 10	$\& \leq 14$	+95	—	+50	+32	—	+16	—	+6	0
> 14	$\& \leq 18$	+95	—	+50	+32	—	+16	—	+6	0
> 18	$\& \leq 24$	+110	—	+65	+40	—	+20	—	+7	0
> 24	$\& \leq 30$	+110	—	+65	+40	—	+20	—	+7	0
> 30	$\& \leq 40$	+120	—	+80	+50	—	+25	—	+9	0
> 40	$\& \leq 50$	+130	—	+80	+50	—	+25	—	+9	0
> 50	$\& \leq 65$	+140	—	+100	+60	—	+30	—	+10	0
> 65	$\& \leq 80$	+150	—	+100	+60	—	+30	—	+10	0
> 80	$\& \leq 100$	+170	—	+120	+72	—	+36	—	+12	0
> 100	$\& \leq 120$	+180	—	+120	+72	—	+36	—	+12	0
> 120	$\& \leq 140$	+200	—	+145	+85	—	+43	—	+14	0
> 140	$\& \leq 160$	+210	—	+145	+85	—	+43	—	+14	0
> 160	$\& \leq 180$	+230	—	+145	+85	—	+43	—	+14	0
> 180	$\& \leq 200$	+240	—	+170	+100	—	+50	—	+15	0
> 200	$\& \leq 225$	+260	—	+170	+100	—	+50	—	+15	0
> 225	$\& \leq 250$	+280	—	+170	+100	—	+50	—	+15	0
> 250	$\& \leq 280$	+300	—	+190	+110	—	+56	—	+17	0
> 280	$\& \leq 315$	+330	—	+190	+110	—	+56	—	+17	0
> 315	$\& \leq 355$	+360	—	+210	+125	—	+62	—	+18	0
> 355	$\& \leq 400$	+400	—	+210	+125	—	+62	—	+18	0
> 400	$\& \leq 450$	+440	—	+230	+135	—	+68	—	+20	0
> 450	$\& \leq 500$	+480	—	+230	+135	—	+68	—	+20	0

Table A.2: ISO tolerance codes showing tolerance margin in μm as a function of nominal dimension (in mm).

Quality	01	0	1	2	3	4	5	6	7
≤ 3	0,3	0,5	0,8	1,2	2	3	4	6	10
$> 3 \quad \& \leq 6$	0,4	0,6	1	1,5	2,5	4	5	8	12
$> 6 \quad \& \leq 10$	0,4	0,6	1	1,5	2,5	4	6	9	15
$> 10 \quad \& \leq 18$	0,5	0,8	1,2	2	3	5	8	11	18
$> 18 \quad \& \leq 30$	0,6	1	1,5	2,5	4	6	9	13	21
$> 30 \quad \& \leq 50$	0,6	1	1,5	2,5	4	7	11	16	25
$> 50 \quad \& \leq 80$	0,8	1,2	2	3	5	8	13	19	30
$> 80 \quad \& \leq 120$	1	1,5	2,5	4	6	10	15	22	35
$> 120 \quad \& \leq 180$	1,2	2	3,5	5	8	12	18	25	40
$> 180 \quad \& \leq 250$	2	3	4,5	7	10	14	20	29	46
$> 250 \quad \& \leq 315$	2,5	4	6	8	12	16	23	32	52
$> 315 \quad \& \leq 400$	3	5	7	9	13	18	25	36	57
$> 400 \quad \& \leq 500$	4	6	8	10	15	20	27	40	63

Quality	8	9	10	11	12	13	14	15	16
≤ 3	14	25	40	60	100	140	250	400	600
$> 3 \quad \& \leq 6$	18	30	48	75	120	180	300	480	750
$> 6 \quad \& \leq 10$	22	36	58	90	150	220	360	580	900
$> 10 \quad \& \leq 18$	27	43	70	110	180	270	430	700	1 100
$> 18 \quad \& \leq 30$	33	52	84	130	210	330	520	840	1 300
$> 30 \quad \& \leq 50$	39	62	100	160	250	390	620	1 000	1 600
$> 50 \quad \& \leq 80$	46	74	120	190	300	460	740	1 200	1 900
$> 80 \quad \& \leq 120$	54	87	140	220	350	540	870	1 400	2 200
$> 120 \quad \& \leq 180$	63	100	160	250	400	630	1 000	1 600	2 500
$> 180 \quad \& \leq 250$	72	115	185	290	460	720	1 150	1 850	2 900
$> 250 \quad \& \leq 315$	81	130	210	320	520	810	1 300	2 100	3 200
$> 315 \quad \& \leq 400$	89	140	230	360	570	890	1 400	2 300	3 600
$> 400 \quad \& \leq 500$	97	155	250	400	630	970	1 550	2 500	4 000

defines the shape of the fit, \varnothing for instance is used for cylindric fits. In the case of cylindric fit, the nominal dimension is the nominal diameter.

For single parts, the dimension and tolerance is expressed by a sequence of shape symbol, nominal dimension, tolerance code. For example $\varnothing 50H7$ defines a referential housing with 50mm nominal dimension while $\varnothing 13g6$ defines a shaft with 13mm nominal dimension.

For assemblies, the symbol and nominal dimension are followed by housing tolerance, then shaft tolerance codes. For example, in Figure 1.2 $\varnothing 69H8f7$ refers to a cylindric loose fit of 69mm nominal diameter.

Considering Table A.1, it is worth noticing that switching tolerance letter of assemblies do not change the fit nature. For instance $H7f7$ defines the same loose fit as $F7h7$, and $H7p6$ defines the same snug fit as $P7h6$.

Appendix B

Dual Vectors

Dual numbers

Dual numbers [53] are defined in a similar way that complex numbers are. In analogy to the imaginary number i that is used to define the imaginary part of a complex number, the dual number ε is used to define the dual part of a dual number. The dual number, however, is defined as non-zero element whose powers are zeros, leading to different arithmetics than imaginary numbers. The dual number is therefor called *nilpotent*.

General dual number ring

Provided $G = \{0, 1, \varepsilon\}$ with multiplication operator \cdot , where

$$\begin{aligned}\varepsilon &\neq 0 \\ \varepsilon^2 &= 0.\end{aligned}$$

we notice that (G, \cdot) is a semigroup with 1 as identity.

Definition B.1 (General dual number ring). A general dual number ring is the semigroup ring [98] of the semigroup G over a ring R [99].

A general dual number is then expressed as $\hat{x} = a + \varepsilon b$ where $a, b \in R$. Addition and multiplication are extended on dual numbers. Given $\hat{x} = x_a + \varepsilon x_b$ and $\hat{y} = y_a + \varepsilon y_b$ where $x_a, x_b, y_a, y_b \in R$, we write:

$$\begin{aligned}\hat{x} + \hat{y} &= (x_a + y_a) + \varepsilon(x_b + y_b) \\ \hat{x} - \hat{y} &= (x_a - y_a) + \varepsilon(x_b - y_b) \\ \hat{x} \times \hat{y} &= (x_a y_a) + \varepsilon(x_a y_b + x_b y_a)\end{aligned}$$

division $\hat{x} = \frac{\hat{a}}{\hat{b}}$ is defined as the solution of the equation $\hat{a} = \hat{b} \times \hat{x}$.

We note that dual scalars are general dual numbers defined over the real numbers ring $(\mathbb{R}, +, \cdot)$.

Dual vectors

When ring R in Definition B.1 is a vector field, the semigroup ring of G over R defines a *dual vector ring*. A dual vector is denoted as $\hat{w} = \vec{v} + \varepsilon \vec{u}$ where $\vec{v}, \vec{u} \in R$.

Dual semifields

Dual numbers can apply not only to rings (and fields) but also to semifields [98].

Definition B.2 (General dual number semi-ring). A general dual number semiring is the semigroup semiring [98] of the semigroup G over a semifield (thus a semiring) R [99].

This extends the use of dual numbers to structures where not every element has a multiplication inverse.

Appendix C

Screw Theory

Mechanical crews

Screw theory [31] studies solid-bodies dynamics and kinematics. In a search of a mathematical tool that can abstract both concepts while still accounting for objects geometry, the theory proposes the use of *screws*, which it defines as follows.

Definition C.1 (Screw). A screw is an ordered 6-tuple. The first triple represents a line Euclidean vector associated, and the second triple represents a free Euclidean vector applied to a point [19].

Each triple represents vector coordinate with respect to a coordinate system $B = \{o, \vec{e}_x, \vec{e}_y, \vec{e}_z\}$, where o is the origin of coordinates, \vec{e}_x , \vec{e}_y and \vec{e}_z are orthogonal right-handed unit vectors. The origin o then defines the point to which the second vector is bound. The screw can then be expressed as follows.

$$\{\vec{v}_1 | \vec{v}_2\} = \left\{ \begin{array}{c|c} v_1^x & v_2^x \\ v_1^y & v_2^y \\ v_1^z & v_2^z \end{array} \right\}$$

The carrier line of the first vector is defined by \vec{v}_1 itself, and a Euclidean vector $\vec{r} = \vec{v}_1 \times \vec{v}_2$ which defines a point on that line.

As mentioned above, screws are abstract mathematical tools. In order to interpret screws, a context, either dynamic or kinematic, should be given.

Twist about a screw

Chasle theory showed that motion of a rigid body between two position can be represented by a translation along an axis, and a rotation about the same axis. The motion of a rigid body can thus be presented by a screw. In this case, the first triple in the screw represents the angular velocity $\vec{\omega}$;

its direction represents the rotation axis, while its magnitude represents the rotation angle per time unit. The second triple represents the linear velocity at the origin \vec{v} . Such a vector is then referred to as *twist*, and written as follows.

$$\mathbf{T} = \{\vec{\omega}|\vec{v}\} = \left\{ \begin{array}{c|c} \omega_x & v_x \\ \omega_y & v_y \\ \omega_z & v_z \end{array} \right\}$$

Wrench on a screw

Poinsot theory showed that the system of external forces acting on a rigid body can be reduced to a force and a torque on plane perpendicular to this force. The system of external forces acting on a rigid body can thus be represented by a screw. In this case, the first triple in the screw represents the force \vec{f} . The second triple represents the torque at the origin \vec{m} . Such a vector is then referred to as *wrench*, and written as follows.

$$\mathbf{W} = \{\vec{f}|\vec{m}\} = \left\{ \begin{array}{c|c} f_x & m_x \\ f_y & m_y \\ f_z & m_z \end{array} \right\}$$

Co-moment of two screws

The co-moment of two screws $\mathbf{A} = \{\vec{a}_1|\vec{a}_2\}$ and $\mathbf{B} = \{\vec{b}_1|\vec{b}_2\}$ is defined as follows:

$$\mathbf{A} \odot \mathbf{B} = \vec{a}_1 \cdot \vec{b}_2 + \vec{b}_1 \cdot \vec{a}_2.$$

When one of the screws is a wrench screw \mathbf{W} , while the other is a twist screw \mathbf{T} , the co-moment of the two screws define their power.

$$\begin{aligned} P &= \mathbf{W} \odot \mathbf{T} \\ &= \vec{f} \cdot \vec{v} + \vec{m} \cdot \vec{\omega}. \end{aligned}$$

We can then state that the work done by a wrench screw \mathbf{W} to displace a rigid body by a twist screw \mathbf{T} during a time interval dt equals to

$$dW = (\mathbf{W} \odot \mathbf{T})dt.$$

Reciprocal screws

Two screws are said to be mutually reciprocal if their virtual coefficient is zero [31]. This means that the work done by a wrench on the first screw to displace a rigid body by a twist about the second one is null over time [134].

Definition C.2 (Reciprocal screws). A pair of wrench screw $W = \{\vec{f}|\vec{t}\}$ and twist screw $T = \{\vec{\omega}|\vec{v}\}$ is reciprocal when the virtual work of the wrench on the twist equals zero.

Screws as dual vectors

Literature suggest the representation of screws, either twists or wrenches, as dual vectors [36, 100]. Thus, a screw $S = \{\vec{v}_1|\vec{v}_2\}$ can be written as

$$S = \vec{v}_1 + \vec{v}_2\varepsilon.$$

In the current work we adopt this convention. This allows the direct application of algebraic properties of dual fields or dual semifields, depending on the underling structure use: rings or semirings respectively.

Appendix D

Description Logic

Decidability

Description Logic (DL) is a family of formal languages that deals with decidable fragments of FOL [17]. It incorporates different languages that vary according to their level of expressiveness. In fact, some DL variants go beyond FOL capacities and define constructs that require higher order logics, they remain however decidable [28]. DL languages exhibit well-understood computational behaviors.

Logic's primitives

DL models the world by means of three logical entities.

Individuals that represent *objects* in the modeled world. Individuals are comparable to constants in FOL.

Concepts that represent objects *sets* in the modeled world. Concepts are comparable to unary predicates in FOL.

Roles that represent objects *relations* in the modeled world. Roles are comparable to binary predicates in FOL.

Sets of primitive concepts and roles that identify a given domain are first defined in a knowledge base. Primitive concepts are referred to as *concept names* and primitive roles as *role names*. Language constructs are used to extend the number of concepts that can be expressed by the language. Newly introduced concepts are *described* using other role names, concept names and *concept descriptions*, by means of language constructs. New roles can also be *described* the same way, and used in concept descriptions.

The set of constructs allowed in a DL language dictates its expressiveness. A kernel set of constructs that is embedded in all DL languages is

Table D.1: A subset of DL constructs, their OWL equivalents, and their semantics in terms of interpretations under the domain of discourse Δ . GCI and axioms' equivalents and interpretations are also shown [17, 28, 179].

DL construct	OWL equivalent	Construct semantic
<i>ALC</i> constructs		
\top	owl:Thing	Δ
\perp	owl:Nothing	\emptyset
$C \sqcap D$	owl:intersectionOf	$C \cap D$
$C \sqcup D$	owl:unionOf	$C \cup D$
$\forall r.C$	owl:allValuesFrom	$\{x, \forall y ((x, y) \in r \Rightarrow y \in C)\}$
$\exists r.C$	owl:someValuesFrom	$\{x, \exists y ((x, y) \in r \wedge y \in C)\}$
$\neg C$	owl:complementOf	Δ/C
Other constructs		
$r \in p$	rdfs:subPropertyOf	$\forall x, y ((x, y) \in r \Rightarrow (x, y) \in p)$
r^-	owl:inverseOf	$\{(x, y), (y, x) \in r\}$
GCI		
$C \sqsubseteq D$	rdfs:subClassOf	$C \subseteq D$
$C \equiv D$	owl:equivalentClass	$C = D$
Axioms		
$C(x)$	rdf:type	$x \in C$
$r(x, y)$:r	$(x, y) \in r$

referred to as *ALC*. In fact, *ALC* is a DL language on its own, that is the simplest among its counterparts.

Table D.1 lists *ALC* constructs, and their respective equivalents in OWL-DL.

Language semantics

DL is credited for its well-defined semantics. Semantics of the logic are defined at the level of linguistic constructs in terms of *interpretations*. An interpretation is possible under a domain of discourse Δ , whenever $\Delta \neq \emptyset$. It maps each concept C to a set $C \subseteq \Delta$, and each role r to a relation $r \subseteq \Delta \times \Delta$. Table D.1 shows the interpretation of each of *ALC* constructs with respect to a domain of discourse Δ .

The TBox \mathcal{T} of a knowledge base is defined in terms of a finite set of General Concept Inclusion (GCI). A GCI is a restriction of the type $C \sqsubseteq D$, where C and D are DL-concepts. A GCI of the type $C \sqsubseteq D$ is interpreted as $C \subseteq D$. A TBox \mathcal{T} is interpreted as the conjunction of the interpretations of all its GCIs.

Note that a concept equivalence, written $C \equiv D$ is a syntactic sugar of

the conjunction of two GCI; $C \sqsubseteq D$ and $D \sqsubseteq C$. If one of the concepts in $C \equiv D$ is a concept name, the statement is then called a *definition*.

The ABox \mathcal{A} of a knowledge base is defined in terms of a finite set of axioms. An axiom can be of the type $C(x)$ where C is a concept and x is an individual. Such an axiom is interpreted as $x \in C$. Alternatively, an axiom can be of the form $r(x, y)$ where r is a role, and x and y are individuals. Such an axiom is interpreted as $(x, y) \in r$. An ABox \mathcal{A} is interpreted as the conjunction of the interpretations of all its axioms.

A knowledge base $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ is interpreted as the conjunction of the interpretations of its TBox and its ABox.

Expressive power

GCI allow for the expression of domain knowledge in term of rules. They can be used to represent concepts hierarchy in the modeled world, e.g. $\text{Student} \sqsubseteq \text{Person}$ that states that all students are people. They can also be used to restrict the domain of a role, e.g. $\exists \text{teachs}.\top \sqsubseteq \text{Teacher}$ that states that only teachers can teach. They can even express more complex rules such as $\exists \text{teachs}.\text{Math} \sqsubseteq \text{Teacher} \sqcap \exists \text{hasDegree}.\text{ScientificDiploma}$ that states that in order to teach math, one has to be a teacher with at least one scientific degree.

We note that expressiveness of the language is highly dependant on the allowed structures. For instance, with \mathcal{ALC} only we cannot put restrictions on roles range. To this end new constructs were added to \mathcal{ALC} , producing new DL languages, at the expense of some computational advantages.

For example, the introduction of the inverse role allows the expression of range restrictions, and much more. Given a role r , an inverse role r^- is defined and interpreted as $r^- = \{(x, y), (y, x) \in r\}$. More constructs are also defined such as transitive roles, subroles, concrete domains and nominals.

A convention exists to name DL languages by adding a letter to the name for each introduced construct. To avoid lengthy names, the core language \mathcal{ALC} augmented by transitive roles is abbreviated as \mathcal{S} . In this work we are particularly interested in the language \mathcal{SHOIQ} , which supports nominals \mathcal{O} , inverse roles \mathcal{I} , and qualified cardinality restriction \mathcal{Q} , besides core constructs \mathcal{S} . A more expressive language variant, \mathcal{SROIQ} [88], is supported by OWL-DL since version 1.1.

Bibliography

- [1] ANR ROMMA project. <http://romma.lmt.ens-cachan.fr>.
- [2] cURL. <http://curl.haxx.se/>.
- [3] Protégé ontology editor. <http://protege.stanford.edu/>.
- [4] TraceParts. <http://www.traceparts.com>.
- [5] World Wide Web Consortium. <http://www.w3.org>.
- [6] Xerces-C++ XML parser. <http://xerces.apache.org/xerces-c/>.
- [7] ISO 10303-1:1994. Industrial automation systems and integration – Product data representation and exchange – Part 1: Overview and fundamental principles. Standard, International Organization for Standardization, Geneva, Switzerland, 1994.
- [8] ISO 10303-203:1994. Industrial automation systems and integration – Product data representation and exchange – Part 203: Application protocol: Configuration controlled 3D design of mechanical parts and assemblies. Standard, International Organization for Standardization, Geneva, Switzerland, 1994.
- [9] ISO 10303-28:1998. Industrial automation systems and integration – Product data representation and exchange – Part 28 - Implementation methods: XML representations of EXPRESS schema and data. Standard, International Organization for Standardization, Geneva, Switzerland, 1998.
- [10] ISO 10303-21:2002. Industrial automation systems and integration – Product data representation and exchange – Part 21: Implementation methods: Clear text encoding of the exchange structure. Standard, International Organization for Standardization, Geneva, Switzerland, 2002.
- [11] ISO 10303-22:2002. Industrial automation systems and integration – Product data representation and exchange – Part 22 - Implementation

- methods: Standard data access interface. Standard, International Organization for Standardization, Geneva, Switzerland, 2002.
- [12] ISO 10303-214:2003. Industrial automation systems and integration – Product data representation and exchange – Part 214: Application protocol: Core data for automotive mechanical design processes. Standard, International Organization for Standardization, Geneva, Switzerland, 2003.
- [13] ISO 10303-11:2004. Industrial automation systems and integration – Product data representation and exchange – Part 11: Description methods: The EXPRESS language reference manual. Standard, International Organization for Standardization, Geneva, Switzerland, 2004.
- [14] OWL Web Ontology Language Overview. <http://www.w3.org/TR/owlfeatures/>, 2004.
- [15] DL Implementation Group. <http://dl.kr.org/dig/>, 2006.
- [16] Technical drawings – Indication of dimensions and tolerances – Part 1: General principles. Standard, International Organization for Standardization, Geneva, Switzerland, 2014.
- [17] ABITEBOUL, S., MANOLESCU, I., RIGAUX, P., ROUSSET, M.-C., SENELLART, P., ET AL. *Web data management*. Cambridge University Press, 2012.
- [18] ACHESON, D. *Elementary Fluid Dynamics*. Oxford Applied Mathematics and Computing Science Series. Oxford University Press, 1990.
- [19] ADAMS, J. D. *Feature based analysis of selective limited motion in assemblies*. PhD thesis, Massachusetts Institute of Technology, 1998.
- [20] AGRAWALA, M., PHAN, D., HEISER, J., HAYMAKER, J., KLINGNER, J., HANRAHAN, P., AND TVERSKY, B. Designing effective step-by-step assembly instructions. In *ACM Transactions on Graphics (TOG)* (2003), vol. 22, ACM, pp. 828–837.
- [21] AIDI, M., TOLLENAERE, M., AND POURROY, F. Towards a numerical simulation scheduling in an engineering system approach. In *Systems, Man and Cybernetics, 2002 IEEE International Conference on* (2002), vol. 4, IEEE.
- [22] ALBERS, A., BRAUN, T., CLARKSON, P. J., ENKLER, H.-G., AND WYNN, D. C. Contact and channel modelling to support early design of technical systems. In *Proc. International Conference on Engineering Design, ICED’09* (Stanford, CA, USA, 2009).

- [23] ALBERS, A., BURKARDT, N., AND OHMER, M. Contact and channel model for pairs of working surfaces. In *Advances in Design*, H. A. ElMaraghy and W. H. ElMaraghy, Eds., Springer Series in Advanced Manufacturing. Springer London, 2006, pp. 511–520.
- [24] ALBERS, A., MATTHIESEN, S., AND LECHNER, G. Konstruktionsmethodisches grundmodell zum zusammenhang von gestalt und funktion technischer systeme. *Konstruktion*, 7-8 (2002), 55–60.
- [25] ALLADA, V., AND ANAND, S. Feature-based modelling approaches for integrated manufacturing: state-of-the-art survey and future research directions. *International Journal of Computer Integrated Manufacturing* 8, 6 (1995), 411–440.
- [26] AMES, A. L. Production ready feature recognition based automatic group technology part coding. In *Proceedings of the First ACM Symposium on Solid Modeling Foundations and CAD/CAM Applications* (New York, NY, USA, 1991), SMA '91, ACM, pp. 161–169.
- [27] BAADER, F., BRANDT, S., AND LUTZ, C. Pushing the \mathcal{EL} envelope. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence* (San Francisco, CA, USA, 2005), IJCAI'05, Morgan Kaufmann Publishers Inc., pp. 364–369.
- [28] BAADER, F., HORROCKS, I., AND SATTLER, U. Chapter 3 description logics. In *Handbook of Knowledge Representation*, V. L. Frank van Harmelen and B. Porter, Eds., vol. 3 of *Foundations of Artificial Intelligence*. Elsevier, 2008, pp. 135 – 179.
- [29] BACON, S. *Reasoning about Mechanical Devices: A Top-down Approach to Deriving Behavior from Structure*. 1988.
- [30] BALDWIN, C., AND CLARK, K. *Design Rules: The power of modularity*. Design Rules. MIT Press, 2000.
- [31] BALL, R. *A treatise on the theory of screws*. Cambridge University University Press, 1900.
- [32] BARBAU, R., KRIMA, S., RACHURI, S., NARAYANAN, A., FIORENTINI, X., FOUFOU, S., AND SRIRAM, R. D. Ontostep: Enriching product model data using ontologies. *Computer-Aided Design* 44, 6 (2012), 575–590.
- [33] BARRETT, C. W., DILL, D. L., AND STUMP, A. Checking satisfiability of first-order formulas by incremental translation to sat. In *Computer Aided Verification* (2002), Springer, pp. 236–249.

- [34] BAYSAL, M. M., ROY, U., SUDARSAN, R., SRIRAM, R. D., AND LYONS, K. The open assembly model for the exchange of assembly and tolerance information: overview and example. In *ASME 2004 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference* (2004), American Society of Mechanical Engineers, pp. 759–770.
- [35] BERNERS-LEE, T., HENDLER, J., LASSILA, O., ET AL. The semantic web. *Scientific american* 284, 5 (2001), 28–37.
- [36] BIDARD, C. Dual basis of screw-vectors for inverse kinestatic problems in robotics. In *Advances in Robot Kinematics and Computational Geometry*, J. Lenarcic and B. Ravani, Eds. Springer Netherlands, 1994, pp. 339–348.
- [37] BLOUNT, G. N., KNEEBONE, S., AND KINGSTON, M. R. Selection of knowledge-based engineering design applications. *Journal of Engineering Design* 6, 1 (1995), 31–38.
- [38] BLÜMEL, E., STRASSBURGER, S., STUREK, R., AND KIMURA, I. Pragmatic approach to apply virtual reality technology in accelerating a product life cycle. In *Proc. of International Conference INNOVATIONS* (Slany, Czech Republic, June 2004), pp. 199–207.
- [39] BOBROW, J. E. Optimal robot plant planning using the minimum-time criterion. *Robotics and Automation, IEEE Journal of* 4, 4 (1988), 443–450.
- [40] BOGART, K., DRYSDALE, S., AND STEIN, C. Discrete math for computer science students, 2004.
- [41] BOUSSUGE, F., LEON, J.-C., HAHMANN, S., AND FINE, L. An analysis of DMU transformation requirements for structural assembly simulations. In *The Eighth International Conference on Engineering Computational Technology* (Dubronik, Croatie, Sep 2012), B.H.V. Topping, Civil Comp Press.
- [42] BOUSSUGE, F., SHAHWAN, A., LÉON, J.-C., HAHMANN, S., FOUCAULT, G., AND FINE, L. Template-based Geometric Transformations of a Functionally Enriched DMU into FE Assembly Models. *Computer-Aided Design and Applications* 11, 04 (2014), 436–449.
- [43] BRIGGS, C., BROWN, G., SIEBENALER, D., FAORO, J., AND ROWE, S. Model based definition, April 2010.
- [44] CANNY, J. F., AND LIN, M. C. An opportunistic global path planner. *Algorithmica* 10, 2-4 (1993), 102–120.

- [45] CARSLAW, H., AND JAEGER, J. *Conduction of Heat in Solids. 2nd edition*. Oxford Science Publications. Clarendon Press, 1986.
- [46] CHAKRABARTI, A., AND BLIGH, T. An approach to functional synthesis of solutions in mechanical conceptual design. part i: Introduction and knowledge representation. *Research in Engineering Design* 6, 3 (1994), 127–141.
- [47] CHAPMAN, C. B., AND PINFOLD, M. Design engineering - a need to rethink the solution using knowledge based engineering. *Knowledge-Based Systems* 12, 5-6 (October 1999), 257–267.
- [48] CHAPMAN, C. B., AND PINFOLD, M. The application of a knowledge based engineering approach to the rapid design and analysis of an automotive structure. *Adv. Eng. Softw.* 32, 12 (Nov. 2001), 903–912.
- [49] CHARDONNET, J.-R., AND LEON, J.-C. Designing interaction in virtual worlds through a passive haptic peripheral. In *2012 IEEE RO-MAN* (France, 2012), pp. 284–289.
- [50] CHOUDRIA, R., AND VÉRON, P. Identifying and re-meshing contact interfaces in a polyhedral assembly for digital mock-up. *Eng. with Computers* 22, 1 (2006), 47–58.
- [51] CHURCH, A. An unsolvable problem of elementary number theory. *American journal of mathematics* 58, 2 (1936), 345–363.
- [52] CLARK, B., HANKS, B., AND ERNST, C. Conformal assembly meshing with tolerant imprinting. In *Proceedings of the 17th international meshing roundtable* (Pittsburg, USA, 2008), Springer, pp. 267–280.
- [53] CLIFFORD, W. Preliminary Sketch of Biquaternions. *Proceedings of The London Mathematical Society* s1-4 (1871), 381–395.
- [54] CONDOOR, S. Integrating design in engineering graphics courses using feature-based, parametric solid modeling. In *Frontiers in Education Conference, 1999. FIE '99. 29th Annual* (1999), vol. 2, pp. 12D2/13–12D2/17.
- [55] CORALLO, A., LAUBACHER, R., MARGHERITA, A., AND TURRISI, G. Enhancing product development through knowledge-based engineering (kbe): A case study in the aerospace industry. *Journal of Manufacturing Technology Management* 20, 8 (2009), 1070–1083.
- [56] CURRAN, R., VERHAGEN, W. J., VAN TOOREN, M. J., AND VAN DER LAAN, T. H. A multidisciplinary implementation methodology for knowledge based engineering: KNOMAD. *Expert Systems with Applications* 37, 11 (2010), 7336–7350.

- [57] DAI, F., AND REINDL, P. Enabling digital mock-up with virtual reality techniques – vision, concept, demonstrator. In *Design Engineering Technical Conferences and Computers in Engineering Conference* (1996).
- [58] DATE, H., KANAI, S., KISHINAMI, T., AND NISHIGAKI, I. Flexible feature and resolution control of triangular meshes. In *Proceedings of the sixth IASTED international conference on visualization, imaging and image processing* (2006).
- [59] DAWOOD, H. *Theories of Interval Arithmetic: Mathematical Foundations and Applications*. Hend Dawood and LAP LAMBERT Academic Publishing, 2011.
- [60] DE KLEER, J. How circuits work. *Artificial Intelligence* 24, 1 (1984), 205–280.
- [61] DE MOURA, L., AND BJØRNER, N. Z3: An efficient smt solver. In *Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 337–340.
- [62] DENG, Y.-M., BRITTON, G., AND TOR, S. A design perspective of mechanical function and its object-oriented representation scheme. *Engineering with Computers* 14, 4 (1998), 309–320.
- [63] DIXON, A., AND SHAH, J. J. Assembly feature tutor and recognition algorithms based on mating face pairs. *Computer-Aided Design and Applications* 7, 3 (2010), 319–333.
- [64] DRIEUX, G., LEON, J.-C., GUILLAUME, F., AND CHEVASSUS, N. Processes to integrate design with downstream applications through product shapes adaptation. *Systems Journal, IEEE* 3, 2 (2009), 199–209.
- [65] EMBEREY, C., MILTON, N., BERENDS, J. P. T. J., TOOREN, M. J. L. V., DER ELST, S. V., AND VERMEULEN, B. Application of knowledge engineering methodologies to support engineering design application development in aerospace. In *7th AIAA Aviation Technology* (Belfast, Northern Ireland, 2007), Integration and Operations Conference (ATIO).
- [66] FALCIDIENO, B., AND GIANNINI, F. Automatic recognition and representation of shape-based features in a geometric modeling system. *Comput. Vision Graph. Image Process.* 48 (October 1989), 93–123.
- [67] FIORENTINI, X., GAMBINO, I., LIANG, V.-C., RACHURI, S., MANI, M., AND BOCK, C. An ontology for assembly representation. Tech.

- rep., National Institute of Standards and Technology NISTIR 7436, Gaithersburg, MD 20899, USA, July, 2007.
- [68] FOUCAULT, G., CUILLIÈRE, J.-C., FRANÇOIS, V., MARANZANA, R., AND LÉON, J.-C. Adaptation of CAD model topology for finite element analysis. *Computer-Aided Design* 40, 2 (2008), 176–196.
- [69] FOUCAULT, G., SHAHWAN, A., LÉON, J.-C., AND FINE, L. What is the content of a DMU? Analysis and proposal of improvements. In *Actes du 12ème Colloque National AIP PRIMECA, Le Mont Dore, 29 Mars – 1er avril 2011 : Produits, Procédés et Systèmes Industriels : intégration Réel-Virtuel* (Le Mont Dore, France, 2011).
- [70] FRENCH, T., HELSEL, J., URBANICK, B., AND SVENSEN, C. *Mechanical Drawing CAD Communications*. Glencoe/McGraw-Hill School Publishing Company, 1990.
- [71] GARBADE, R., AND DOLEZAL, W. DMU@Airbus – Evolution of the Digital Mock-up (DMU) at Airbus to the Centre of Aircraft Development. In *The Future of Product Development*, F.-L. Krause, Ed. Springer Berlin Heidelberg, 2007, pp. 3–12.
- [72] GERO, J. S. Design prototypes: a knowledge representation schema for design. *AI Mag.* 11, 4 (Oct. 1990), 26–36.
- [73] GERO, J. S., AND KANNENGIESSER, U. The situated function-behaviour. *Design Studies* 25 (2004), 373–391.
- [74] GOPALAKRISHNAN, P. *Handbook Of Materials Management*. Eastern economy edition. Prentice-Hall Of India Pvt. Limited, 2004.
- [75] GRIMM, S., AND MOTIK, B. Closed world reasoning in the semantic web through epistemic operators. In *OWL: Experiences and Directions* (2005).
- [76] GRUBER, T. R. A translation approach to portable ontology specifications. *Knowledge acquisition* 5, 2 (1993), 199–220.
- [77] GUI, J.-K., AND MÄNTYLÄ, M. New concepts for complete product assembly modeling. In *Proceedings on the second ACM symposium on Solid modeling and applications* (New York, NY, USA, 1993), SMA '93, ACM, pp. 397–406.
- [78] HAARSLEV, V., AND MÖLLER, R. Racer: A Core Inference Engine for the Semantic Web. In *Workshop on Evaluation of Ontology-based Tools* (2003).

- [79] HAGHIGHI, K., AND KANG, E. A knowledge-based approach to the adaptive finite element analysis. In *Modeling, Mesh Generation, and Adaptive Numerical Methods for Partial Differential Equations*. Springer, 1995, pp. 267–276.
- [80] HAMRI, O., LÉON, J.-C., GIANNINI, F., FALCIDIENO, B., POULAT, A., AND FINE, L. Interfacing product views through a mixed shape representation. part 1: Data structures and operators. *International Journal on Interactive Design and Manufacturing (IJIDeM)* 2, 2 (2008), 69–85.
- [81] HAN, J., PRATT, M., AND REGLI, W. C. Manufacturing feature recognition from solid models: A status report. *IEEE Transactions on Robotics and Automation* 16 (2000), 782–796.
- [82] HAN, S. M., BENAROYA, H., AND WEI, T. Dynamics of transversely vibrating beams using four engineering theories. *Journal of Sound and Vibration* 225, 5 (1999), 935 – 988.
- [83] HARTENBERG, R., AND DENAVIT, J. *Kinematic synthesis of linkages*. McGraw-Hill series in mechanical engineering. McGraw-Hill, 1964, ch. Concepts and Notations Related to Mechanisms, pp. 28–67.
- [84] HASHIM, F. M., JUSTER, N., AND PENNINGTON, A. A functional approach to redesign. *Engineering with Computers* 10, 3 (1994), 125–139.
- [85] HENDERSON, M. R., AND TAYLOR, L. E. A meta-model for mechanical products based upon the mechanical design process. *Research in Engineering Design* 5, 3-4 (1993), 140–160.
- [86] HIRTZ, J., STONE, R. B., MCADAMS, D. A., SZYKMAN, S., AND WOOD, K. L. A functional basis for engineering design: reconciling and evolving previous efforts. *Research in Engineering Design* 13, 2 (2002), 65–82.
- [87] HORRIDGE, M., AND BECHHOFFER, S. The OWL API: A java API for OWL ontologies. *Semantic Web* 2, 1 (2011), 11–21.
- [88] HORROCKS, I., KUTZ, O., AND SATTLER, U. The even more irresistible *SRIOQ*. *KR* 6 (2006), 57–67.
- [89] IACOB, R., MITROUCHEV, P., AND LÉON, J.-C. Contact identification for assembly–disassembly simulation with a haptic device. *The Visual Computer* 24, 11 (2008), 973–979.
- [90] IYER, N., JAYANTI, S., LOU, K., KALYANARAMAN, Y., AND RAMANI, K. Three-dimensional shape searching: state-of-the-art review and future trends. *Computer-Aided Design* 37, 5 (2005), 509–530.

- [91] JACKSON, D. Automating first-order relational logic. In *ACM SIGSOFT Software Engineering Notes* (2000), vol. 25, ACM, pp. 130–139.
- [92] JONES, M., PRICE, M., AND BUTLIN, G. Geometry management support for auto-meshing. In *Proceedings of 4th International Meshing Roundtable* (1995), pp. 153–164.
- [93] JORISSEN, P., AND LAMOTTE, W. A framework supporting general object interactions for dynamic virtual worlds. In *Smart Graphics*, A. Butz, A. Krüger, and P. Olivier, Eds., vol. 3031 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2004, pp. 154–158.
- [94] JOSHI, S., AND CHANG, T. C. Graph-based heuristics for recognition of machined features from a 3d solid model. *Comput. Aided Des.* 20 (March 1988), 58–66.
- [95] JOURDES, F., BONNEAU, G.-P., HAHMANN, S., LÉON, J.-C., AND FAURE, F. Computation of components’ interfaces in highly complex assemblies. *Computer-Aided Design* 46 (2014), 170–178.
- [96] KALLMANN, M., AND THALMANN, D. Direct 3d interaction with smart objects. In *Proceedings of the ACM symposium on Virtual reality software and technology* (New York, NY, USA, 1999), VRST ’99, ACM, pp. 124–130.
- [97] KALMAN, J. Bednarek’s extension of light’s associativity test. *Semi-group Forum* 3, 1 (1971), 275–276.
- [98] KANDASAMY, W. *Smarandache Semirings, Semifields, and Semivector Spaces*. American Research Press, 2002.
- [99] KANDASAMY, W., AND SMARANDACHE, F. *Dual Numbers*. Zip Publishing, 2012.
- [100] KELER, M. L. On the theory of screws and the dual method. In *Proceedings of A Symposium Commemorating the Legacy, Works, and Life of Sir Robert Stawell Ball Upon the 100th Anniversary of A Treatise on the Theory of Screws* (2000), University of Cambridge.
- [101] KHATIB, O. Real-time obstacle avoidance for manipulators and mobile robots. *The international journal of robotics research* 5, 1 (1986), 90–98.
- [102] KIM, K.-Y., MANLEY, D. G., AND YANG, H. Ontology-based assembly design and information sharing for collaborative product development. *Computer-Aided Design* 38, 12 (2006), 1233–1250.

- [103] KIM, K.-Y., WANG, Y., MUOGBOH, O. S., AND NNAJI, B. O. Design formalism for collaborative assembly design. *Computer-Aided Design* 36, 9 (2004), 849 – 871.
- [104] KITAMURA, Y., AND MIZOGUCHI, R. An ontological schema for sharing conceptual engineering knowledge. In *International workshop on semantic web foundations and application technologies* (2003), pp. 25–28.
- [105] KUTTIG, D. Potential and limits of functional modelling in the cad process. *Research in Engineering Design* 5, 1 (1993), 40–48.
- [106] KYPRIANOU, L. K. *Shape classification in computer-aided design*. PhD thesis, University of Cambridge, 1980.
- [107] LEE, J., AND LAI, K.-Y. What’s in design rationale? *Human–Computer Interaction* 6, 3-4 (1991), 251–280.
- [108] LEIZEROWICZ, W., BILGIC, T., LIN, J., AND FOX, M. S. Collaborative design using www. In *Proc. of WET-ICE’96* (1996), University of West Virginia.
- [109] LEON, J. C., AND FINE, L. A new approach to the preparation of models for fe analyses. *Int. J. of Computer Applications in Technology* 23 (2005), 166–184.
- [110] LEVISON, L., AND BADLER, N. I. How animated agents perform tasks: connecting planning and manipulation through object-specific reasoning. In *AAAI 1994 Spring Symposium Series. Proceedings* (1994), ScholarlyCommons, Penn Engineering.
- [111] LI, K. *Shape Analysis of B-Rep CAD Models to Extract Partial and Global Symmetries*. PhD thesis, Grenoble University, Grenoble, November 2011.
- [112] LIANG, V.-C., AND PAREDIS, C. J. A port ontology for conceptual design of systems. *Journal of Computing and Information Science in Engineering* 4, 3 (2004), 206–217.
- [113] LIN, M. C., AND CANNY, J. F. A fast algorithm for incremental distance calculation. In *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on* (1991), IEEE, pp. 1008–1014.
- [114] LOU, R., PERNOT, J.-P., MIKCHEVITCH, A., AND VÉRON, P. Merging enriched finite element triangle meshes for fast prototyping of alternate solutions in the context of industrial maintenance. *Computer-Aided Design* 42, 8 (2010), 670–681.

- [115] LOVETT, P., INGRAM, A., AND BANCROFT, C. Knowledge-based engineering for SMEs – a methodology. *Journal of Materials Processing Technology* 107, 1–3 (2000), 384 – 389.
- [116] LUTZ, C. *The Complexity of Reasoning with Concrete Domains*. PhD thesis, RWTH Aachen, 2001.
- [117] MAKEM, J. E., ARMSTRONG, C. G., AND ROBINSON, T. T. Automatic decomposition and efficient semi-structured meshing of complex solids. In *Proc. of the 20th international meshing roundtable*. Springer, 2012, pp. 199–215.
- [118] MANDIL, G. *Modèle de représentation géométrique intégrant les états physiques du produit*. PhD thesis, Université de Sherbrooke, Quebec, 2012.
- [119] MANTYLA, M. Topological analysis of polygon meshes. *Computer-Aided Design* 15, 4 (1983), 228–234.
- [120] MÄNTYLÄ, M. *An Introduction to Solid Modeling*. Principles of computer science series. Computer Science Press, 1988.
- [121] MÄNTYLÄ, M., NAU, D., AND SHAH, J. Challenges in feature-based manufacturing research. *Commun. ACM* 39, 2 (Feb. 1996), 77–85.
- [122] MARR, D., AND NISHIHARA, H. K. Representation and recognition of the spatial organization of three-dimensional shapes. *Proceedings of the Royal Society of London. Series B. Biological Sciences* 200, 1140 (1978), 269–294.
- [123] MATHESON, J. A. L. *Hyperstatic Structures: An Introduction to the Theory of Statically Indeterminate Structures*, vol. 1 of *Hyperstatic Structures: An Introduction to the Theory of Statically Indeterminate Structures*. Butterworths, 1971.
- [124] MCMAHON, C., AND BROWNE, J. *CADCAM: Principles, Practice and Manufacturing Management*. ADDISON WESLEY Publishing Company Incorporated, 1998.
- [125] MITRA, N. J., YANG, Y.-L., YAN, D.-M., LI, W., AND AGRAWALA, M. Illustrating how mechanical assemblies work. vol. 29, ACM, p. 58.
- [126] MOKHTARIAN, F., AND MACKWORTH, A. K. A theory of multiscale, curvature-based shape representation for planar curves. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 14, 8 (1992), 789–805.
- [127] OHSUGA, S. Toward intelligent CAD systems. *Computer-aided design* 21, 5 (1989), 315–337.

- [128] ORTON, J. D., AND WEICK, K. E. Loosely coupled systems: A reconceptualization. *Academy of Management Review* 15, 2 (1990), 203–223.
- [129] PAHL, G., BEITZ, W., AND WALLACE, K. *Engineering Design: Systematic Approach*. Springer-Verlag GmbH, 1996.
- [130] PIEGL, L. A., AND TILLER, W. *The NURBS book*. Springer, 1997.
- [131] POINCARÉ, H. Analysis situs. *Journal de l'École polytechnique* 11 (1895).
- [132] POINSOT, L. *Eléments de statique*. Mallet-Bachelier, 1861.
- [133] POMMIER, S., AND BERTHAUD, Y. *Mécanique Générale*. In [134], 2010, ch. Actions, Liaisons, pp. 83–112.
- [134] POMMIER, S., AND BERTHAUD, Y. *Mécanique Générale*. Dunod, Paris, 2010.
- [135] POOLE, D. L., AND MACKWORTH, A. K. *Artificial Intelligence: foundations of computational agents*. Cambridge University Press, 2010.
- [136] QIAN, L., AND GERO, J. S. Function-behavior-structure paths and their role in analogy-based design. *AI EDAM* 10, 4 (1996), 289–312.
- [137] QUADROS, W. R., AND OWEN, S. J. Defeaturing cad models using a geometry-based size field and facet-based reduction operators. *Engineering with Computers* 28, 3 (2012), 211–224.
- [138] QUINLAN, S. Efficient distance computation between non-convex objects. In *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on* (1994), IEEE, pp. 3324–3329.
- [139] QUINLAN, S., AND KHATIB, O. Elastic bands: Connecting path planning and control. In *Robotics and Automation, 1993. Proceedings., 1993 IEEE International Conference on* (1993), IEEE, pp. 802–807.
- [140] QUINTANA, V., RIVEST, L., PELLERIN, R., VENNE, F., AND KHEDDOUCI, F. Will model-based definition replace engineering drawings throughout the product lifecycle? a global perspective from aerospace industry. *Computers in Industry* 61, 5 (2010), 497 – 508.
- [141] RAGHAVAN, V., MOLINEROS, J., AND SHARMA, R. Interactive evaluation of assembly sequences using augmented reality. *IEEE Trans. Robot. Autom.* 15, 3 (1999), 435–449.
- [142] RAHMANI, K., AND THOMSON, V. Ontology based interface design and control methodology for collaborative product development. *Comput. Aided Des.* 44, 5 (May 2012), 432–444.

- [143] ROCCA, G. L. Knowledge based engineering: Between AI and CAD. Review of a language based technology to support engineering design. *Advanced Engineering Informatics* 26, 2 (2012), 159 – 179.
- [144] ROCCA, G. L., AND TOOREN, M. J. L. V. Enabling distributed multi-disciplinary design of complex products: a knowledge based engineering approach. *Journal of Design Research* 5, 3 (2007), 333–352.
- [145] ROCCA, G. L., AND VAN TOOREN, M. A knowledge based engineering approach to support automatic generation of fe models in aircraft design. In *45th AIAA Aerospace Sciences Meeting and Exhibit* (2007).
- [146] ROY, U., AND BHARADWAJ, B. Design with part behaviors: behavior model, representation and applications. *Computer-Aided Design* 34, 9 (2002), 613 – 636.
- [147] ROY, U., PRAMANIK, N., SUDARSAN, R., SRIRAM, R., AND LYONS, K. Function-to-form mapping: model, representation and applications in design synthesis. *Computer-Aided Design* 33, 10 (2001), 699 – 719.
- [148] RUSS, B., DABBEERU, M. M., CHORNEY, A. S., AND GUPTA, S. K. Automated assembly model simplification for finite element analysis. In *ASME 2012 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference* (2012), American Society of Mechanical Engineers, pp. 197–206.
- [149] SAS, O. C. Open CASCADE Technology, 3D modeling & numerical simulation. <http://www.opencascade.org>.
- [150] SCHENK, M., STRASSBURGER, S., AND KISSNER, H. Combining virtual reality and assembly simulation for production planning and worker qualification. In *Proc. of International Conference on Changeable, Agile, Reconfigurable and Virtual Production* (Munich, Germany, 2005).
- [151] SEDERBERG, T. W. *Computer Aided Geometric Design*. Brigham Young University, 2011, ch. Tensor-product Surfaces.
- [152] SHAH, J., AND MÄNTYLÄ, M. *Parametric and Feature-Based CAD/CAM: Concepts, Techniques, and Applications*. A Wiley-Interscience publication. Wiley, 1995.
- [153] SHAH, J., SREEVALSAN, P., AND MATHEW, A. Survey of CAD/feature-based process planning and NC programming techniques. *Computer-Aided Engineering Journal* 8, 1 (Feb 1991), 25–33.
- [154] SHAH, J. J., ANDERSON, D., KIM, Y. S., AND JOSHI, S. A discourse on geometric feature recognition from cad models. *Journal of Computing and Information Science in Engineering* 1, 1 (2001), 41–51.

- [155] SHEARER, R., MOTIK, B., AND HORROCKS, I. Hermit: A highly-efficient OWL reasoner. In *OWLED* (2008), vol. 432.
- [156] SIRIN, E., PARSIA, B., GRAU, B. C., KALYANPUR, A., AND KATZ, Y. Pellet: A practical OWL-DL reasoner. *Web Semantics* 5, 2 (June 2007), 51–53.
- [157] SMITHERS, T. AI-based versus geometry-based design or why design cannot be supported by geometry alone. *Comput. Aided Des.* 21, 3 (Apr. 1989), 141–150.
- [158] SONTI, R., KUNJUR, G., AND GADH, R. Shape feature determination using the curvature region representation. In *Proceedings of the fourth ACM symposium on Solid modeling and applications* (1997), ACM, pp. 285–296.
- [159] SRIDHARAN, N. *Classification, Parameterization, and Recognition of NC Machining Features with Sculptured Surfaces*. Arizona State University, 2000.
- [160] SRINIVASAN, V., LOVEJOY, W. S., AND BEACH, D. Integrated product design for marketability and manufacturing. *Journal of Marketing Research* (1997), 154–163.
- [161] STOKES, M., AND CONSORTIUM, M. *Managing Engineering Knowledge: MOKA: Methodology for Knowledge Based Engineering Applications*. Professional Engineering Publishing Limited, 2001.
- [162] STURGES, R. H., O'SHAUGHNESSY, K., AND KILANI, M. I. Computational model for conceptual design based on extended function logic. *Artificial Intelligence for Engineering, Design, Analysis and Manufacturing* 10, 04 (1996), 255–274.
- [163] SUH, N. *The Principles of Design*. Oxford series on advanced manufacturing. Oxford University Press on Demand, 1990.
- [164] SUNIL, V., AGARWAL, R., AND PANDE, S. An approach to recognize interacting features from b-rep cad models of prismatic machined parts using a hybrid (graph and rule based) technique. *Computers in Industry* 61, 7 (2010), 686–701.
- [165] SWAMIDASS, P. *Encyclopedia of Production and Manufacturing Management*. Springer, 2000.
- [166] THAKUR, A., BANERJEE, A. G., AND GUPTA, S. K. A survey of CAD model simplification techniques for physics-based simulation applications. *Computer-Aided Design* 41, 2 (2009), 65–80.

-
- [167] TOMIYAMA, T., UMEDA, Y., AND YOSHIKAWA, H. A CAD for functional design. *CIRP Annals - Manufacturing Technology* 42, 1 (1993), 143 – 146.
- [168] TROUSSIER, N., POURROY, F., AND TOLLENAERE, M. Mechanical models management in engineering design. In *IDMME* (France, 1998), vol. 4, pp. 1087–1094.
- [169] TSARKOV, D., AND HORROCKS, I. Fact++ description logic reasoner: System description. In *In Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2006)* (2006), Springer, pp. 292–297.
- [170] TURING, A. M. On computable numbers, with an application to the entscheidungsproblem. *J. of Math* 58 (1936), 345–363.
- [171] ULLMAN, D. *The Mechanical Design Process*. McGraw-Hill series in mechanical engineering. McGraw-Hill, 2003.
- [172] ULLMAN, D. G. A taxonomy for mechanical design. *Research in Engineering Design* 3, 3 (1992), 179–189.
- [173] ULLMAN, D. G. *The mechanical design process*. 2nd ed. New York: McGraw-Hill, 1997.
- [174] UMEDA, Y., TAKEDA, H., TOMIYAMA, T., AND YOSHIKAWA, H. Function, behaviour, and structure. *Applications of artificial intelligence in engineering V 1* (1990), 177–194.
- [175] VENKATARAMAN, S., AND SOHONI, M. Reconstruction of feature volumes and feature suppression. In *Proceedings of the seventh ACM symposium on Solid modeling and applications* (2002), ACM, pp. 60–71.
- [176] VERHAGEN, W. J., BERMELL-GARCIA, P., VAN DIJK, R. E., AND CURRAN, R. A critical review of knowledge-based engineering: An identification of research challenges. *Advanced Engineering Informatics* 26, 1 (2012), 5 – 15. Network and Supply Chain System Integration for Mass Customization and Sustainable Behavior.
- [177] WANG, G. G. Definition and review of virtual prototyping. *Journal of Computing and Information Science in Engineering (Transactions of the ASME)* 2, 3 (2002), 232–236.
- [178] WELCH, R., AND DIXON, J. Guiding conceptual design through behavioral reasoning. *Research in Engineering Design* 6, 3 (1994), 169–188.

- [179] WELTY, C., MCGUINNESS, D. L., AND SMITH, M. K. Owl web ontology language guide. *W3C recommendation, W3C (February 2004)* <http://www.w3.org/TR/2004/REC-owl-guide-20040210> (2004).
- [180] WHITNEY, D. Designing the design process. *Research in Engineering Design* 2, 1 (1990), 3–13.
- [181] WHITNEY, D. *Mechanical Assemblies: Their Design, Manufacture, and Role in Product Development*. No. v. 1 in Mechanical Assemblies: Their Design, Manufacture, and Role in Product Development. Oxford University Press, 2004.
- [182] XU, X., AND HINDUJA, S. Recognition of rough machining features in 2.5D components. *Computer-Aided Design* 30, 7 (1998), 503–516.
- [183] Y14.5, A. *Geometric Dimensioning & Tolerancing*. ASME, 1982.
- [184] ZHANG, Q., VONDEREMBSE, M. A., AND CAO, M. Product concept and prototype flexibility in manufacturing: Implications for customer satisfaction. *European Journal of Operational Research* 194, 1 (2009), 143 – 154.
- [185] ZHU, H., AND MENQ, C. B-Rep model simplification by automatic fillet/round suppressing for efficient automatic feature recognition. *Computer-Aided Design* 34, 2 (2002), 109–123.