Functional Feature Modeling in Mechanical Product Development

by

Zhengrong Cheng

A thesis submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Engineering Management

Department of Mechanical Engineering
University of Alberta

Abstract

Engineering design is a complex process. There are unknowns and uncertainties. To better support engineering design, researches on different aspects of engineering design need to be conducted. Function concepts represent the engineering design purposes of the product elements and are usually considered to be interdependent from the detailed geometric solutions. Unfortunately, functional considerations are often not represented systematically and explicitly in Computer-aided Design (CAD). The gaps between functional and geometrical representations of the design models require an extensive research to unveil their associations and interactions.

The main objective of the thesis is to incorporate functional design capability with CAD. It begins with a quantitative analysis of design dependencies among design element at the product design level, which indicates that there exist mixed types of design dependencies, both functional and structural. Next, with the dependency analysis of CAD models, it shows that CAD modeling at best complies with the form of the design at the current stage. The current practice of CAD modeling lacks functional design flavor. Because functional design is critical in the engineering design process, it justifies the need of extending CAD modeling capability to support functional design. Functional feature, which inherits from the associative feature, is introduced to address the issue of connectivity deficiency between the declarative functions in high-level design and procedural feature operations in detailed CAD modeling. A methodology for CAD modeling that is driven by the functions of the design artifact with functional features has been proposed. Functions, as the result of traditional functional design, are taken as function attributes in the functional feature. Abstract geometry features are proposed as functional concept carriers. It shows how to construct abstract geometry features and how to synthesize them in the detailed

CAD models. The research results of this thesis provide concepts and tools for designers, CAD practitioners, and researchers to better understand the design dependencies and carry out function-oriented modeling.

Preface

Unless otherwise stated, this thesis is the result of my own research under the supervision of Prof. Yongsheng Ma. Any reference to other people's work is clearly cited in the text. This thesis has not been submitted in whole or in part for consideration of any other degree or qualification at this or any other university or institution. Some of the work contained in this dissertation has been published in the journals and/or conference proceedings. See below for the list of publications.

- Cheng Z.R., Ma Y.S. (2017) "A Functional Feature Modeling Method" *Advanced Engineering Informatics,* 33 1-15. doi:10.1016/j.aei.2017.04.003.
- Cheng Z.R., Ma Y.S. (2017) "Explicit function-based design modeling methodology with features", *Journal of Engineering Design,* 28(3), pp 205-231. doi: 10.1080/09544828.2017.1291920.
- Cheng Z.R., Ma Y.S. (2017) "Graph Centrality Analysis of Feature Dependencies to Unveil Modeling Intents", accepted to *Computer-aided Design and Application*, Taylor & Francis. doi: 10.1080/16864360.2018.1441236.
- Cheng Z.R., Ma Y.S. A network-based assessment of design dependency in product development, in *Innovative Design and Manufacturing (ICIDM), Proceedings of the 2014 International Conference*, pp.137-142, 13-15 Aug. 2014 doi: 10.1109/IDAM.2014.6912684.
- Cheng Z.R., Tang S.H, Chen G., Ma Y.S. (2013) "Unified Feature Paradigm". *Semantic Modeling and Interoperability in Product and Process Engineering* pp 117-142. Springer Series in Advanced Manufacturing, London: Springer

- Liu J.K., Cheng Z.R., Ma Y.S. (2016) “Product design-optimization integration via associative optimization feature modeling”, *Advanced Engineering Informatics,* 30, 713–727. doi:10.1016/j.aei.2016.09.004.

Other than the introduction in Chapter 1 and literature review in Chapter 2, Chapter 3 of this thesis is mainly based on a conference paper: Cheng Z.R., Ma Y.S. A network-based assessment of design dependency in product development, in *Innovative Design and Manufacturing (ICIDM), Proceedings of the 2014 International Conference*, pp.137-142, 13-15 Aug. 2014 doi: 10.1109/IDAM.2014.6912684. This chapter, which mainly focuses on the analysis of products with assembly structure, lays the groundwork for the research in network analysis in the part level with the nodes representing features. The analysis with feature dependency graph has been accepted to a Conference proceeding, CAD 2017. Its extension has been accepted to the journal *Computer-aided Design and Application*, the main contents of which could be found in Chapter 4 of the thesis. These two chapters apply network analysis approach to assess dependencies among design elements at different levels and show that, on the one hand, design dependencies are complex and include multiple types of dependencies, and, on the other hand, CAD modeling procedures at the current stage focus on the structural aspects of the design.

Starting from Chapter 5, we start to dig into the function level of engineering design with CAD. Chapter 5 is mainly based on a journal publication: Cheng Z.R., Ma Y.S. (2017) “Explicit function-based design modeling methodology with features”, *Journal of Engineering Design* 28(3), pp 205-231. It is an extension of my supervisor’s associative feature concept. Chapter 6 is the detailed description of one of the key elements in functional feature, abstract geometry feature, with examples. It is the result of a journal publication: Cheng Z.R., Ma Y.S. (2017), "*A*

Functional Feature Modeling Method" *Advanced Engineering Informatics,* 33 1-15. doi:10.1016/j.aei.2017.04.003. Chapter 7 contains some of the content from the paper published in *Journal of Engineering Design* and unpublished works. The slotted liner case study in Chapter 8 is based on the partial result of the paper published in *Journal of Engineering Design*. The other case studies have not been published.

Acknowledgements

First, I would like to thank my supervisor Dr. Yongsheng Ma for his kind guidance, supports, cares and encouragements during my PhD study. Without his help, this work would not be possible.

I would also like to acknowledge my supervisory committee members, Dr. Ahmed Qureshi and Dr. Hossein Rouhani, who have provided invaluable suggestions to improve my research work. In addition, thanks to examiners Dr. Alexandra Komrakova and Dr. Andrew Martin, who have asked very constructive questions and offered valuable suggestions in the candidacy exam.

I would like to express my gratitude to China Scholarship Council (CSC), Mitacs, Canada Pump and Power (CPP), Reservoir Management Inc., and NSERC for their financial support.

My thanks also go to my colleagues in Dr. Ma's research group, who contribute to a friendly research environment. They are more than colleagues, they are friends and families.

Last but not least, I appreciate the support from my family. Although my parents didn't have much education, they understand the importance of it and they've supported me throughout the journey. To my wife, Jingdi Wen, your love is my spring of courage. Thank you for bringing Elaina into our world.

Contents

List of Tables

List of Figures

1 Introduction

1.1 Background

Engineering design is a complex process that involves a lot of unknowns and uncertainties. One of the main contributions to this phenomenon is the complicated relations among different design elements in different levels of details with various representations. Multiple efforts on engineering design focus on different aspects of it, trying to make engineering design more effective and efficient. For example, from Function, Behavior, and Structure (FBS), Axiomatic Design (AD), and Requirement Functional Logical Physical (RFLP) that center on the conceptual level of design to explicit reference modeling, horizontal modeling, and resilient modeling that focus on the detailed modeling in CAD (Umeda et al. 1990; Suh 2001; Gebhard 2013; Landers and Khurana 2004; Bodein, Rose, and Caillaud 2014, Dassault Systèmes 2017[1]). In the conceptual function design, according to Pahl et al. (2007) a function is "fulfilled by the physical effects", which is realized by the working surface. In general, it is possible that different structures can be used to satisfy a specified function.

Computer-aided Design (CAD) has been indispensable in the modern engineering design practice. From 2D sketch to 3D wireframe, 3D solid, and feature-based parametric CAD, engineers find it more convenient to model the product shape. Instead of manipulating lower level geometric entities to produce the desired product geometry, engineers with the help of modern CAD tools could model products with features that encapsulate geometric elements with

[1] Dassault Systèmes (2017), System Architecture Design & Simulation. Retrieved from https://www.3ds.com/industries/high-tech/smarter-faster-lighter/systems-architecture-design-simulation/

engineering semantics, adding a layer of abstraction. Feature-based CAD systems facilitate the modeling of design artifacts.

However, not all of the created CAD models are well reusable. Even with visually the same resulting CAD geometry, the modeling procedures and operations applied might be quite diverse. Without a standard way of model construction, it is hard to modify CAD models to cater to new design requirements. In some cases, even a single alteration of a certain value in the model could render the whole part unusable, which is even worse if it is not visually identifiable. The reason behind it, the author believe, is that the non-optimal and implicit modeling strategy applied to create the model. To reach a more robust CAD modeling strategy, a better understanding of the nature of the CAD model construction is necessary.

The author believes the key lies in the understanding of modeling intents. Modeling intents are what behind the CAD model construction, i.e., what users wish the model to be. There are two levels of modeling intents, i.e., the reasons why models are constructed in certain ways to, firstly, conform to the physical structure, and secondly, comply with functional design considerations. Understanding modeling intents is critical. If changes are about to be made to the model, it is better to know how and why the model has been constructed in certain ways such that when the changes are carried out the model will at least not blow up. If the intents of model construction are unknown, it would be difficult to change the model properly due to its complex inner parametric and geometric associations. Moreover, if the modeling intents are revealed, engineers can see whether the model has been constructed robustly by judging, for example, if the functional considerations have been conformed to.

There is still a gap between engineers' mental model of the product and the CAD modeling operations, which is unintuitive and makes it difficult to follow design changes due to the interleaving dependencies among the feature operations. Feature dependencies include datum dependency, geometry dependency, and parameter dependency. For example, when a sketch references a previously defined datum, the changing position of the datum will carry the sketch around, which incurs datum dependency. The discrepancy between the mental model of the designer, which is functional, and the modeling procedure used in CAD, which is procedural, forces users to have separate flows of thoughts and to maintain the associativity by checking the consistency of constraints involved in the engineering design processes. In addition, due to the intricate dependencies and the lack of management tools, it is tedious and error-prone to maintain the functional and geometrical integrity of the design.

It is argued that the future success of CAD will hinge on its ability to support design engineers in more advanced aspects other than replicating the geometry of the design artifacts in the forms of points, lines, surfaces, and/or volume primitives only (Schulte et al., 1993). Akman, Hagen, and Tomiyamat (1990) argued that CAD system did not support a crucial ingredient of design, i.e., interaction with intelligence. Yasushi Umeda and Tomiyama (1997) mentioned that future CAD technology should represent and reason about function and argued that traditional CAD lack of those capabilities. Unfortunately, current CAD is still not intelligent enough and it falls short of the functional design support.

1.2 Research scope and objectives

The main objective of the current thesis is to incorporate functional design capability into CAD. It begins with a quantitative analysis of design dependencies among design element in the

product design level, which indicates that there exist mixed types of design dependencies, both functional and structural. Next, with dependency analysis of CAD models, it shows that in CAD modeling at best complies with the form of the design at the current stage. The current practice of CAD modeling lacks functional design flavor. Because functional design is critical in the engineering design process, it justifies the need for extending CAD modeling capability to support functional design. This research extends the associative feature proposed in (Ma et al. 2007) to address the issue of the lack of connectivity between function and design in CAD modeling. To achieve this goal, this research proposes a methodology for CAD modeling that is driven by the functions of the design artifact with functional features. The research scope and objectives are briefly presented as follows.

- Assessment of design dependencies among design elements at the product level in a global manner

Some complex products consist of many different parts with complicated assembly structures. The interactions among design elements are intricate. For example, given a product structure, how are different design elements related, and which design elements are more critical than others? Increasing the human understanding of the interactions among design elements, be them physical quantities or virtual ones, is beneficial. This subtopic proposes a global measure to analyze the graph of the product structure. The result of this research subtopic is useful in resource allocation in engineering design and engineering change management. It is observed that design contains different types of dependencies, including both functional and structural dependencies. It will be shown that the CAD model construction process mainly focus on the structural aspect.

- Feature dependencies assessment in CAD models

Other than assembly level product structure, the interactions of design elements in the part-level, which are the features in CAD, also need to be examined. This subtopic moves into the design with CAD modeling in the part level. In modern CAD, a part consists of multiple features that users have applied to build up the digital representation of the physical part. The procedure of applying different features creates feature dependencies. By analyzing feature dependencies insights of the underlining modeling intents can be gained. This subtopic applies different centrality measures to analyze the resulting graph from the feature dependencies. It is revealed that CAD modeling at the current stage is form-centric. Compared with the result of the above-mentioned subtopic, it shows that CAD modeling needs a function-integrated approach.

- A flexible knowledge representation to associate geometries with different levels of details (LOD).

It is observed that design process is evolving with enriching geometries of different levels of details. Current existing CAD modeling approaches do not consider this critical aspect. Since geometries with different LODs represent different embodiments of design concepts, the associations among them are evident due to their evolvements in different stages of the design process. They capture some fundamental functional design considerations of the design. However, this type of association is not well supported in the current CAD system and modeling methodology. This subtopic will provide a representation for geometry with a lower level of detail and discuss how they evolve with design and how they can be used as functional concept carriers.

- A generic representation scheme to incorporate functional design knowledge

Engineering knowledge, including properties, behaviors, shapes, and interrelationships among objects, as well as causal information relating objects through physical phenomena, either quantitatively or qualitatively, of the design artifacts should be systematically represented and made available for effective use in the CAD systems. Functional modeling capability should be added to CAD systems to link the lower level geometric descriptions of the design to engineers' mental model through. This subtopic proposes a new type of feature, functional feature, to address this issue. Functional features can capture engineering knowledge and functional design rationale with necessary geometries to manifest such functional considerations in the CAD systems.

It is noted that different manufacturing processes impose different constraints on the design, which result in some variations of the final design shapes. These variations in the shapes of the design reflect different functional considerations for the chosen manufacturing processes, Note that manufacturing process related considerations will not be discussed extensively in this thesis. This does not restrict the application of this research result in that domain.

In addition, there are also researches like Functionally Graded Material (FGM) and Topology Optimization (TO) that are not our target application fields. One can find concepts like function or functional in those domains. However, their characteristics of functions or functionals and the way to achieve them are different. For example, TO usually is achieved by minimizing a mathematical concept, an objective functional, e.g., the elastic strain energy, with certain constraints. For another example, FGM is a topic of material science, where the function is achieved by the variation in composition and microstructure from one material to another with a specific gradient.

1.3 Roadmap of the thesis

This thesis is prepared following the guideline from the Faculty of Graduate Studies and Research (FGSR) at the University of Alberta. It consists of 8 Chapters. This chapter introduces the general context of engineering design with computer-aided tools, the existing problems in the current industry practice, and the scope and road map of the thesis. The rest of the thesis is organized as follows.

- Chapter 2 provides a literature review to current research, including feature technology, CAD modeling methodologies, communication of design intents in CAD, function modeling in engineering design, and the application of graph theory in product design.
- Chapter 3 applies the graph approach to design dependency assessment at the macroscopic level, i.e., the assembly structure. Two concepts, design domination and subordination weights, are introduced to highlight the result of design dependencies in the product structure, the dependencies of which are not restricted to the type of node. The design dependencies could be functional, structural, or any other kinds. The case study from a literature demonstrates that dependencies are mixed in engineering design.
- Chapter 4 takes a step further to use the graph approach to examine CAD models in the part level. Feature information is extracted and organized in the adjacency list representation of the graph. Visualization of the feature dependencies is achieved. Centrality analysis is carried out to find the critical features in the model construction. The result indicates that the current CAD modeling is geometry-centric, i.e., how to build up the form of the design, without embedding functional design considerations.
- Chapter 5 introduces the functional feature framework to bridge the gap between CAD modeling and functional design. The semantics of the functional feature definition is

provided. The ingredients of functional feature, including physics feature, abstract geometry feature, constraints, and parameterization are introduced to lay the foundations for functional feature modeling.

- Chapter 6 applies the abstract geometry feature in the product model construction. The concept of abstract geometry feature has been briefly discussed in Chapter 5. This chapter is devoted to it because the concept is critical in the functional feature modeling framework. A general procedure, starting from functional design to detailed CAD modeling, is introduced. An example is demonstrated based on a commonly seen mechanical product.
- Chapter 7 discusses the function aspect of applying functional feature in CAD. Functional feature implementation considerations in different levels of product design are provided, including part level, module level, and product level. It shows that functions can be abstract or concrete. Functional decomposition is needed to break down a higher-level function to lower-level ones.
- Chapter 8 applied the proposed functional feature in three case studies. These case studies are used to demonstrate the functional feature concept in action. They each focus on different aspects of design. The case study of slotted liner demonstrates the application of functional feature in the part modeling. The case study of road crossing and engine block show that functional feature modeling approach works well in the assembly design and the configuration design with assembly structures. Some CAD tools with GUI have been developed to support the design process.
- The last chapter summarizes the thesis with observations and discussions. Possible future works are also provided.

2 Literature Review

2.1 Chapter introduction

The previous chapter gives a basic introduction to the thesis work. To lead the course of this research towards the objectives identified in the previous chapter, an extensive review of the current literature is necessary. This chapter reviews five aspects of the related literature, including feature technology in section 2.2, CAD modeling and methodology in section 2.3, communication of design intents in section 2.4, functional modeling in engineering design in section 2.5, and the application of graph in engineering design in section 2.6.

2.2 Feature technology

According to Shah and Mantyla (1995), features represent the engineering meanings or significances of the geometry of a part or an assembly and can serve as building blocks for product definition and geometric reasoning. Manufacturing planning gives the origin for feature technologies where features correspond to the volumes in the product that can be machined with a single or a sequence of operations (Berg, Bronsvoort, and Vergeest 2002) e.g., hole features, instead of cylinders, such that features can have engineering semantics.

Feature technologies have been applied to CAD in great extent. Features encapsulate certain geometric and topological entities with engineering semantics. Current mainstream CAD systems provide parametric modeling operations in terms of features, e.g., block feature, extrude features, revolve features, and fillet features, etc., which are generally categorized as form features. Figure 1 provides an example mechanism for parametric feature-based CAD. Assembly features, expressing the relationships that exist between different parts within an assembly, are

applied to the position or orient of the parts, restraining the degree of freedoms, usually in the form of mating conditions (Murshed et al. 2007). Modern part modeling in CAD is a process of creating product geometry with a series of feature operations to construct product shape digitally without much concern of the internal representation of the CAD geometry. The procedure involves the generation of direct 3D primitives or indirect 3D shapes through operations like extruding or sweeping 2D shapes, as well as *Boolean* operations to add or remove volumetric entities.

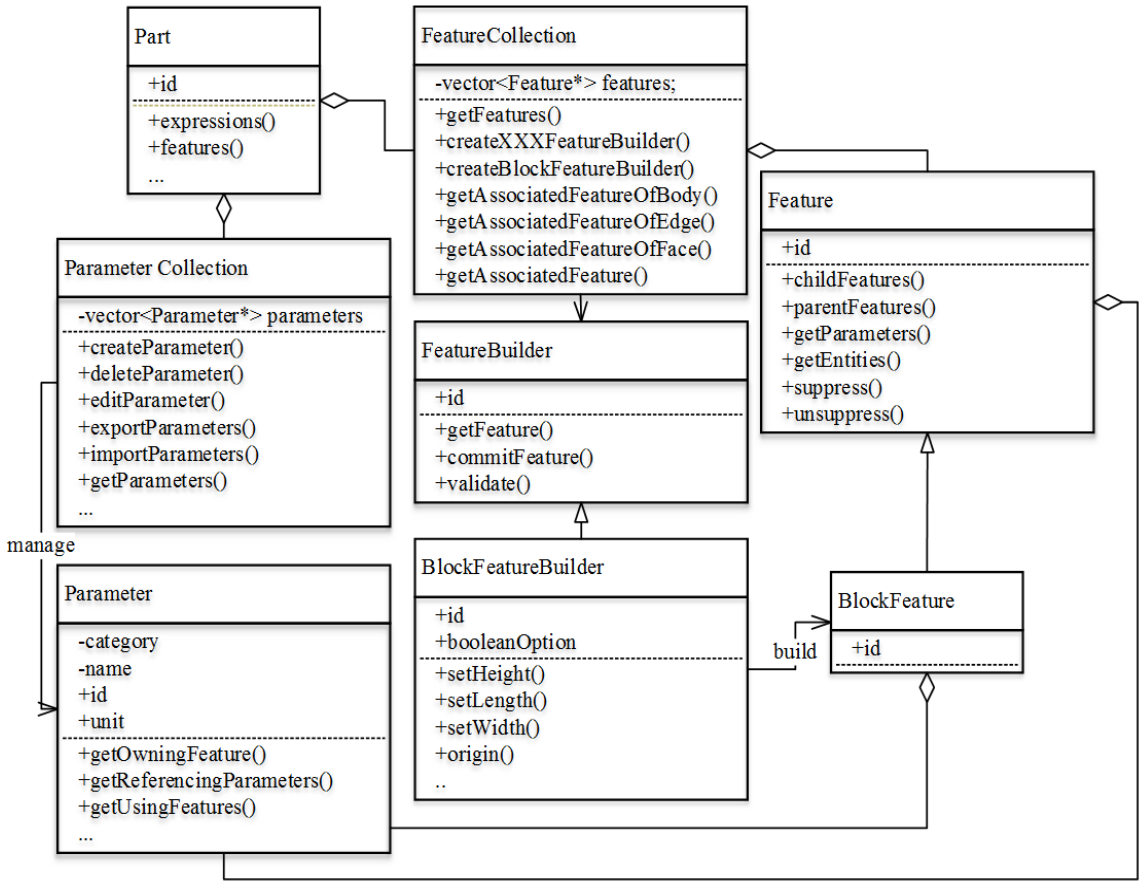


Figure 1 Parametric feature-based CAD

Some CAD system support assembly features, for example, *Catia*® from Dassault Systèmes. One cannot create assembly features between two geometrical elements belonging to

the same component. Assembly features in *Catia*® can only be created between the child components of the active (sub-) assembly. Supported assembly features in *Catia*® include split, hole, hole series, and pocket, etc. For example, assembly hole allows users to create holes passing through multiple parts. Although such holes can be created in each individually, assembly features make it easier. Apparently, this definition of assembly features is different from the one containing assembly constraints.

Other features have been created to extend their application domains, for example, user-defined features (Hoffmann and Joan-Arinyo 1998), associative features (Ma and Tong 2003; Ma et al. 2007), kinematics features (Aifaoui et al. 2006), rib features (Li et al. 2014), and user-defined freeform feature (He et al. 2014). Techniques like user-defined features (UDF) (Hoffmann and Joan-Arinyo 1998; He et al. 2014) can be defined and applied in the model creation to enrich the feature availability. In Ma's associative assembly design feature (Ma et al. 2007) they represent the geometric patterns across multiple parts. It has the characteristics of built-in associative links to the involved geometric entities, self-validation for consistency checking, and other interfacing capabilities. It has the capability to interface with the information defined across multiple part models. From the case study, it can be found that assembly design feature in Ma et al. (2007) is used as the controlling elements. For example, it defines parameters that are inherited to the component level. It defines reference entities, both geometric and non-geometric ones, that can be used in both part and assembly design levels.

Feature parameter maps are generalized dependency maps among different parameters (Yin and Ma 2012). In Yin and Ma (2012), feature parameter map is a conceptual organization scheme for modeling dependencies among parameters at a lower level of information granularity than features. The procedure follows a top-down design approach. Excel was used for the

implementation where parameters relations are embedded into formulas. A set of parameters designed to be interfacing with CAD parametric models in part or assembly levels are maintained specifically such that expression synchronization mechanism available in the CAD tool can be used to update the CAD model from Excel. Different levels of feature parameter maps can be constructed, for example, at the conceptual design level, component design level and assembly design level.

Table 1 presents a partial list of feature types from the literature. As is pointed out in a recent review by Sanfilippo and Borgo (2016), since different feature types tend to be application dependent, a shared methodology for feature classification is non-existent. However, based on the object-oriented software engineering methodology suggested by Ma (2013), feature modeling provides a high-level interface that permits declarative specification and clustering of entities in terms of geometric and dimensional constraints with references to appropriate geometric entities created with a solid modeling engine. Therefore, features can provide enriched semantics for product modeling.

Table 1 Features types found in the literature

Feature types	Comment	Sources
Form features	A set of geometric entities with engineering or functional related specifications	Liu and Nnaji 1991; Shah and Rogers 1988
Assembly features	Expressing the relationships that exist between different parts within an assembly	Murshed et al. 2007
Associative features	Dealing with intricate geometrical relations	Ma and Tong 2003; Ma et al., 2007; Ma 2013
Unified features	Defining the generic common characteristics (attributes and methods) of application features	Chen et al. 2004; Chen et al. 2005; Ma and Hadi 2012
Kinematics features	Used to study motion of artifacts	Aifaoui et al. 2006
Manufacturing feature	Related to manufacturing/machining process	Li et al. 2014; Hoque et al. 2013
Function feature	Providing info on function and role of area in a designed part	Ando et al. 2010
Functional feature	Related to functions of design	Schulte et al. 1993; Myung and Han 2001

2.3 CAD modeling methodologies

Other than the elementary entities, such as points, lines, and faces, solid representations are commonly used in CAD to represent a geometric object, including the interior shape, such as a cube, and a cylinder. There are mainly two commonly used solid representations in CAD, namely, Boundary representation, or B-rep, and Constructive Solid Geometry, or CSG. B-rep extends the wireframe model by adding face information, where a solid is bounded by its surface and has its interior and exterior (Patrikalakis and Maekwa 2003). A CSG solid is constructed from a few primitives, for example, a block, triangular prism, sphere, cylinder, cone, and torus, with *Boolean* operations, i.e., union, intersection, and difference (Patrikalakis and Maekwa 2003). Some CAD software integrates both approaches and supports the history tracking of geometric construction of the model, such as the history mode in Siemens NX®. A general feature-based CAD modeling procedure is shown in Figure 2.

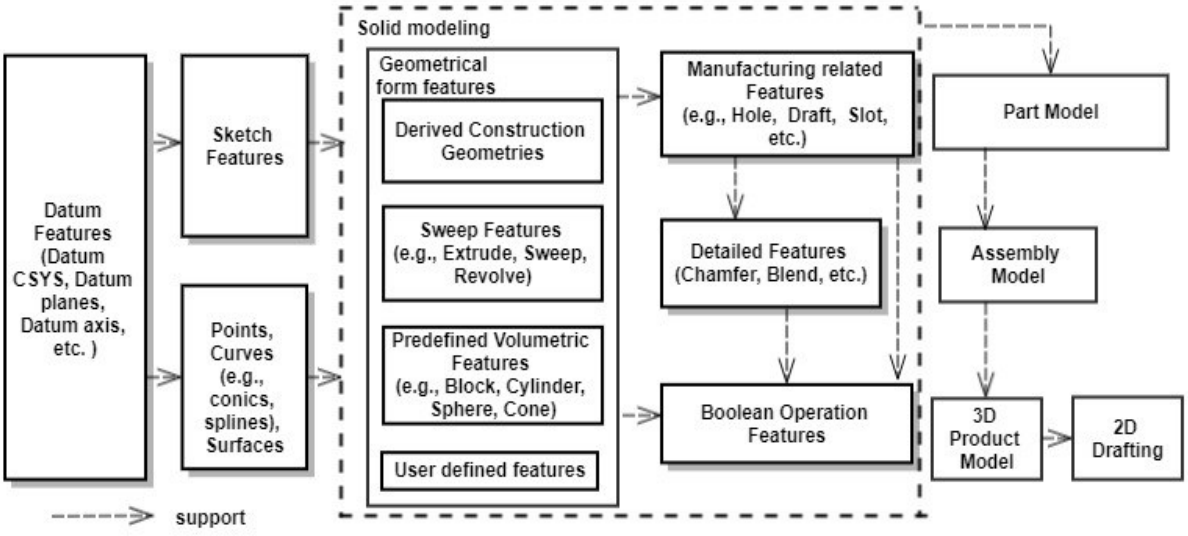


Figure 2 A general feature-based CAD modeling procedure

The parametric capability of feature-based CAD makes it easier to integrate quantitative design knowledge into the model such that it is possible to change the product model with the alterations of values. Dimensions of each feature are controlled by a set of parameters. Since product models are created with features, if applied properly, the whole product shape can be manipulated with a set of parameters. Lin and Hsu (2008) presented an automated design system for drawing dies built on top of a commercial CAD system in a knowledge-based approach. By using the combined capacity of CAD system, design formulas and geometric operations of modeling processes are generated by the system with a minimum set of structure parameters to reduce design time. However, the formulation and determination of the parameters are unclear.

Since feature creation might depend on some features applied previously, feature dependencies are resulted (Bidarra and Bronsvoort 2000). The internal tree structure for features in history-based CAD helps to keep the associated (i.e. parent/children) relations among the features. Powerful as the systems are, the burden of choosing appropriate parameterization and feature operation sequences are still loaded on the designers. Although the parametric design has been applied in the CAD modeling, existing literature acknowledges the management of structure parameters, i.e., geometrically related parameters, without considering management of non-geometric parameters that have an impact on the product geometry in the CAD system.

CAD modeling strategies, for example, explicit reference modeling, resilient modeling, and horizontal modeling, are available to improve the efficiency of model construction. Generally speaking, those CAD modeling methodologies focuses solely on how to improve the efficiency of CAD model construction operationally. For example, horizontal modeling (Landers and Khurana 2004) tries to eliminate the parent/child dependencies between features by creating a bunch of datum planes after a base feature from which following children features are attached.

The resulting feature tree is horizontally structured without long-chained feature dependencies. However, design intents are hard to express in their feature trees. In addition, technically users cannot apply horizontal modeling without a patent license from Delphi.

Similarly, with understanding the problem of unstable CAD models, resilient modeling strategy (Gebhard 2013) offers a solution by managing the sequence and structure of the feature tree. It aims to maximize the flexibility and robustness of CAD models while minimizing inconsistencies by defining a collection of best practices where features are organized in six sequential groups according to their importance, function, and volatility. The six feature groups are reference features, construction features, core features, detail features, modify features, and quarantine features. This approach tries to manipulate the feature tree according to the shape of the design artifact.

Bodein et al. (2013) presented a framework of actions that can guide designers to improve CAD efficiency by utilizing the advantages of parametric CAD in the automotive industry. Their CAD strategy roadmap consists of standardization, advanced methodology, KBE, and expert rules check, where much training is required. Bodein et al. (2014) proposed a practical method for complex part modeling in parametric CAD system by an explicit management of references. They decompose a part model into different regions by their "functions" and model those geometries individually at first with their own references. Later, *Boolean* operations are applied to those "functional geometries". Note that in their research their "functional geometries" are solids that can be applied with Boolean operations. It is not clear whether their approach is still applicable when a part region has overlapping or disjoint functions. Moreover, no details of how to reach the functional geometries of the design are given.

Based on the above discussions it can be seen that both horizontal modeling and resilient modeling strategies focus on managing feature operations to avoid the over-complex feature dependencies with unstable feature tree without considering the functional design considerations. It can be deduced that CAD modeling without following any particular methodology tends to be ad hoc, constructing the shape without considering the editability and function representation of the design. Although explicit reference modeling considers functions of the design to some extent, implementation-wise they use solids to represent functions and apply only Boolean operations on them, which is restrictive. As mentioned above, it is unclear what to do if a region represents multiple functions, and vice versa.

CAD modeling is not just about creating product geometry. It has been integrated into broader frameworks. Table 2 provides some approaches for engineering design with CAD. RFLP is a model-based system engineering approach that takes CAD models as one of the "Physical" (the "P" in RFLP) representations, i.e., the virtual solution. CAD models are the components of system engineering process, assuming CAD models provide a valid virtual representation without looking into the details of how to construct CAD models. Knowledgeware from Dassault Systems supports parameter control with formulas. It is also capable of the rule-based reasoning (*if-else-then*) and checking for constraint validation. Users can create intelligent and automated templates from a model. Knowledge Fusion from Siemens NX® is a generative modeling language based on the principle of KBE. Geometric objects and pertaining operations can be carried out by Knowledge Fusion program with system and user classes.

Table 2 Selected approaches toward engineering design related to CAD

Approaches	Comments	Source
RFLP	Industrial implementation of system engineering approach	Dassault Systèmes (2017)
Explicit reference, Horizontal modeling, Resilient modeling	CAD modeling methodologies focusing on reusability	Bodein et al. (2014); Gebhard (2013); Landers and Khurana 2004
Knowledgeware, Knowledge Fusion	Industrial implementation of Knowledge-based engineering	Dassault Systèmes (2017); Siemens NX (2017)[2]; Amadori et al. 2012
Top-down modeling	Assembly design, KBE	Gao et al. 2013; Amadori et al. (2012)

2.4 Communication of design intents through CAD

Another aspect of building robust CAD models is to foster effective communication of design intents, or design rationale, through CAD models. Note that there is no consensus on the definition of design intent as researchers suggest their own definitions (Iyer and Mills 2006). Generally, design intents are what behind a design decision, i.e., the "why" part of the design. Design intents need to be documented and managed (Elgh 2014). In CAD, design intent can be reflected in how the model is constructed through proper constraints and parameterization, and in how it behaves, for example, when dimensions are modified. They are usually expressed

[2] Siemens NX. (2017), NX Knowledge Fusion for Designers, https://www.plm.automation.siemens.com/nl_nl/support/trainingen/overzicht/nx/nx-knowledge-fusion-designers.shtml

implicitly and approaches like annotations have been applied to encourage the communication of design intents (Camba et al. 2014; Camba and Contero 2015). Annotation elements, such as comments, are commonly used for clarification or explanation (Camba and Contero 2015). For example, in the programming process of software development, comments are used to clarify the functionality and sometimes to provide implementation explanation of the piece of code.

In the engineering design domain, annotations are usually seen in 2D drawings, which provide necessary information like dimensions and tolerances for the purpose of manufacturing, as well as complementary verbal explanations that cannot be shown in the dimensioning of the product. In the 3D CAD environment, technologies like Product and Manufacturing Information (PMI) provide an approach to attach different information to a part or assembly model. PMI objects include dimensions, datum, notes, symbols, and section views, to be used by downstream activities, for example, tooling, manufacturing, inspection, and shipping. Moreover, PMI objects support necessary operations on themselves, for instance, move, edit, delete, and control of visibility.

In academia, Camba et al. (2014) presented an extended annotation method to communicate geometric design intents where design information is represented both internally within the CAD model and externally on a separate repository. Semantic annotations of CAD models are also available in the research community. For example, a system based on segmentations (Attene et al. 2009) of 3D surface meshes and annotations of the detected part shapes expressed by ontology has been reported. Shapes are decomposed into interesting features within the multi-segmentation framework and annotation pipelines are used to attach semantics to the features and the whole shape. However, as Attene et al. (2009) admitted, the inference capability of the system is limited.

Although annotation approach helps to convey design intents in CAD models between designers, users can not interact with the CAD model directly through annotations. Moreover, when the CAD model is changed, the annotation might not be updated fully, automatically, and correctly. In addition, even though other designers understand the design intents behind the CAD model with the help of annotations, it is still not clear about: (1) how to effectively alter the CAD model to cope with new design requirements; (2) whether the model will update smoothly with the alterations. The annotation is functionally separated from the model construction. Hence the approach of applying annotations on the CAD model to communicate design intents is not enough.

It would be superior if the created models themselves reflect design intents and are responsive to the changes of design intents. Design intents are usually aligned with the functionalities of the design artifact, which means that if the CAD models can be constructed in a functional way they can convey design intents systematically. Since the construction approach is associated with the features representing functional intents, they are easier to be altered when functional changes are made.

2.5 Functional modeling in engineering design

Functional modeling serves as a means of linking the different levels of product or system design, which is conceptual. However, there is a huge challenge to unify different definitions and representations of functions in engineering design from literature (Hamraz et al. 2015). Rodenacker (1971) defined a function as a relationship between input and output of energy, material, and information, which is widely accepted in design research (Erden et al. 2008). Function in Miles (1972) is represented as "verb + noun", which shares subjectivity to some extent. For example, Keuneke (1991) classifies function types into ToMake, ToMaintain,

ToPrevent, and ToControl. In the function-behavior-state (FBS) (Y. Umeda et al. 1990; Yasushi Umeda et al. 2005; Tetsuo Tomiyama 1994), a function is defined as "a description of behavior". In the Theory of Technical Systems (Hubka and Eder 1988), functions are defined as the abilities of the technical system to fulfill a purpose. According to Pahl et al. (2007) a function is "fulfilled by the physical effects", i.e., the working geometry, the arrangement of working surfaces. The working surfaces are determined by the type, shape, position, size, and number (Pahl et al. 2007).

Gero (1990) used functions to embody the expectations or purposes of the design artifact into design descriptions. Deng et al. (1998) categorized functions into three main viewpoints, namely, system viewpoint, performance viewpoint, and designer viewpoint. The system viewpoint sees function as a relationship between the input, output, and the state variables of a system; the performance viewpoint perceives function as an abstraction of physical behavior, the set of which defines a functional class with the results as its function; and the designer viewpoint, from the perspective of teleology, views a function as a description of design intent. Deng et al., (1998) also identified four different types of mechanical functions, i.e., assembly function, manufacturing function, marketing function, and maintenance function. Chandrasekaran and Josephson (2000) modeled function as effects in a device-centric view, i.e. an intended or desired role an artifact plays in its environment.

Teoh and Case (2004) introduced the concept of a functional diagram to represent function and structure interaction, which is a network connected by multiple function units. Two objects, function operator and function operand, are linked by a function, which takes the form of a verb or verb phrase that defines the action, to form a function unit in the form of function operator - function -> function operand. In (Goel et al. 2009), a function is defined as a schema that specifies its preconditions and post-conditions in their structure, behavior, and function model.

Their structure is represented in terms of the components and behavior as a sequence of, and transitions between, the states. Ontologies have also been applied to describe functions (Borgo et al. 2011; Gruber 1995; Kitamura and Mizoguchi 2003; Kitamura et al. 2004).

Functional decomposition (Pahl et al. 2007) is applied to break down an overly abstract function into several more specific primitive functions, which are usually called sub-functions. A function structure is the compatible combination of sub-functions into an overall function (Pahl et al. 2007). Umeda et al. (1996) categorized functional decomposition into task decomposition and causal decomposition. Task decomposition is explicitly related to functional knowledge and maintained manually as a mental simulation activity, the results of which are sub-functions that are not causally related. Causal decomposition requires the knowledge of physical behavior and results in causally related sub-functions. An example of a functional decomposition of the assembly of connectors can be found in Deng (2013). Yuan et al. (2016) proposed a hybrid approach to automate functional decomposition in conceptual design, where qualitative processing reasoning, physical effect decomposition, and, if applicable when no physical effect is found for a given function, backward search decomposition are applied. However, their approach bears some limitations, e.g., not considering static behaviors of the system, and that the finest level of decomposed functions corresponds to product components. Nevertheless, the idea behind their approach is generic.

Functions are not stand alone in the engineering design. Instead, they are interrelated to other aspects of the engineering design at the system level. Researches like Quality Function Deployment (QFD), Axiomatic Design (AD), and Design Structure Matrix (DSM) have been applied to correlate the functions or functional requirements with aspects like customer requirements, design parameters, product structures, design tasks, etc., where a form of matrix is

applied to map the relations among any two of those aspects (Suh 2001; Eppinger and Browning 2012; Torres et al. 2010). Usually the values in the matrix are either binary (0 or 1), indicating the existence of correlation, or some real values to reflect the strength of the interrelations, probability or likelihoods of change propagation or impacts, and extents of dependency of the elements from two domains or two elements of the same domain, e.g., customer needs and design characteristics, function and structure (Ahmad et al. 2013; Torres et al. 2010; Eppinger and Browning 2012). A good review of design theory with function supports could be found in (T. Tomiyama et al. 2009).

2.6 Graph theory in product design

Graph, or network, has been used to describe product structures with different representations. For example, Design Structure Matrix (DSM), in the form of a matrix representation of a system or a project, has been applied in product and process design (Eppinger and Browning 2012). MacCormack et al. (2006) proposed propagation cost and clustered cost based on Design Structure Matrix (DSM) to compare the structures of different software designs. The propagation cost assumes equal values among both direct and indirect dependencies, whereas clustered cost assigns different cost across cluster or module. Le et al. (2014) applied graph to model the structure and evolution of products. One of the case studies they used is a physical product and they took the parts as nodes and physical connection as edges of the graph. The evolution of the product is captured by adding and removing of nodes and edges. A few network measures were employed to quantify the evolutionary characteristics of product structures, for example, the average degree, the degree distribution, the density, the clustering coefficient, and the average shortest path, etc.

Jaiswal, Huang, and Rai (2014) introduced an assembly-based conceptual 3D modeling with unlabeled components, where a probabilistic factor graph was used to encapsulate the relationships between the unlabeled components in a shape database. Cheng and Chu (2012) applied a network approach to assess the impacts of changes on a complex product, where a product is considered as a weighted network of parts, subassemblies, or subsystems. Three changeability indices, including degree-changeability, reach-changeability, and between-changeability were presented. However, the dependency relationship between parts is documented with approaches like interviewing experienced engineers, which is subjective and time-consuming.

Researches of applying graph theory in CAD model similarity analysis and retrieval for reuse have been widely found (Tao et al. 2012; Li et al. 2009). Basically, graphs are used to represent the CAD models and the similarity between two 3D CAD models can be evaluated with graph matching algorithms. The graphs can be constructed using different elements of the CAD model. Some construct the graph from the shape of the CAD models with B-rep. For example, Tao et al. (2012) use a representation of Face Attributed Relational Graph (ARG), where faces are taken as nodes and edges connecting the faces as arcs in the graph, created from a B-rep CAD model to convert partial retrieval problem into a subgraph matching problem. Later, in Tao, Wang, and Chen (2015) they constructed a Face Adjacency Graph (FAG) from B-rep models and assessed the model similarity by segmenting the graph into a set of region graphs for subgraph matching.

On the other hand, Li et al. (2009) describe a reuse-oriented retrieval method for CAD models where modeling knowledge are captured in model similarity assessment with a feature dependency directed acyclic graph and subgraph decomposition, i.e., their graphs were not

constructed based on the shape of the CAD model but their construction features. The resulting graph was used to simplify the CAD models such that shape histogram can be constructed based on the simplified CAD models.

3 Dependency Assessment in Product Development

3.1 Chapter introduction

Design dependencies are the kind of dependencies among design elements that are incurred by the existence of information exchange among them. A design element is a general term to denote a design object of the product. A design element can be a part, a feature, or a (sub-) assembly. It is believed that a better understanding of design dependencies among design elements is critical in product design and development. Design dependency analysis in the design of complex product helps designers to make better strategic engineering decisions, e.g., avoiding unnecessary design changes, allocating design resources, and scheduling design activity, especially in the redesign of an existing product.

This chapter proposed a novel approach to identify and rank the critical design elements in the system from the perspective of design dependency analysis. The critical components of a system are defined as 1 those design elements hold dominant positions in the design dependency graph, the design decision or design changes of which have big influences over others, 2 those design elements subject to the design of other elements. Different design strategies can be carried out resulting from the assessment of design dependency. A case study is presented to validate the proposed methods. It is about to show from the case study that design dependencies are of mixed type. The merit of the proposed design dependency assessment method is that it takes a global approach to assess dependency on the whole product.

3.2 The need of analyzing design dependency

People start to pay attention to the importance of understanding design dependencies from engineering changes propagation. Design dependencies provide the means and paths for engineering change propagations. In terms of the definition of engineering changes, the research community has not reached a consensus yet (Huang et al. 2003; Wright 1997; Rouibah and Caskey 2003). For example, Rouibah and Caskey (2003) defined engineering change as "change or modifications in the form, representation, design, material, dimensions, functions, etc., of a product or component after an initial engineering decision has been made". According to Eckert et al. (2004) engineering change could be divided into initiated changes and emergent changes. Mehta et al. (2013) proposed a knowledge-based approach to determine important engineering change (EC) attributes, which are formulated as a multi-objective optimization problem, for engineering change evaluation. They considered two target tasks. One is to determine similar ECs and predict the impacts of the proposed ECs. The other is to maximize the two measures. The goal of their research is to evaluate and divide EC impacts into either high or low.

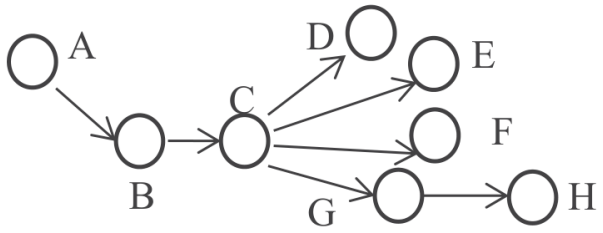


Figure 3 Example to show the categorization of changes (Eckert et al. 2004)

From the change propagation pattern characteristics, engineering changes can also be defined in four categories (Eckert et al. 2004):

- Constants, those components that are unaffected by changes

- Absorbers, which absorb more changes than they cause
- Carriers, which absorb and cause a similar number of changes, and
- Multipliers that generate more changes than they absorb.

For example, in Figure 3 A is a constant; B and G are carriers; C is a multiplier; and D, E, F, and H are absorbers. Their classification of engineering changes increases the understanding of change propagation behavior and design dependencies among design elements. However, this categorization of change propagation behavior is local in the sense that for each node it only cares about its own out-degree and in-degree without a bigger picture of the whole product. For example, although B and G in Figure 3 are both defined as carriers, their real rules in the change propagation are different. The change in B will propagate to C, which will further propagate to many others, whereas the change in G can only propagate to H and not further.

A better understanding of design dependencies of different design elements is critical in the evaluation of engineering change propagation because a change in one design element most likely propagate to others along the path in design dependency graph. A DEPNET solution was proposed in Ouertani (2008) to capture product specification dependencies with a dependency network, which is an oriented graph consisting of nodes that represent product specification and arcs that denote dependency relationships between these specifications. Note that their approach deals with specifications that are already generated and known in the design process.

3.3 Design dominator and subordinator

3.3.1 Overview

The approach in this chapter investigates the behavior of design elements under design dependency analysis in a global manner. The proposed method assesses the design elements based on their positions in the design dependency graph to see which ones are dominant and which are subordinate in a quantitative manner. The design dependency graph is a digraph with nodes denoting the design elements and arcs denoting the dependency. The direction is pointing from the design element that exerts the influence on the one that subjects to the influence. The dependency denotes the effect of the change or design decision in one design element on the other, which might be structural or functional. A design dependency matrix can be used to represent the graph.

Before discussing the method proposed in this article, it is necessary to look at two concepts suggested in Kleinberg (1999) and Easley and Kleinberg (2010):

- Authorities: webpages that pointed to by highly ranked webpages
- Hubs: webpages that point to highly ranked webpages.

In the domain of engineering design, a similar concept can be applied with some adjustment. Here is the proposition of defining design dominator and design subordinator.

- Design dominator: a design element that holds a dominant position in design dependency graph, the design decisions or changes made of which exert influences on others.

- Design subordinator: a design element that subordinates to the design of other elements because the design of such elements is easily influenced by those "dominators" defined above.

Identification of design dominators and subordinators are critical for a better understanding of system behavior of design elements in product development. Another way to understand the concept of design dominator and subordinator is to see how the design changes propagate due to design dependencies. For example, design dominators are the main contributors to propagate the knock-on effects of the design changes to other parts of the product, whereas the design subordinators are subject to the design decisions or design changes made by the design dominators.

3.3.2 Design domination weight and design subordination weight

It is not an either-white-or-black scenario for the two concepts introduced above. A design element can be dominator and subordinator at the same time, however, with different degree of extent. For example, in Figure 4 node 1 to 4 are design dominators, as they have effects on node 5 and others; node 7 to 10 are subordinators, e.g., they receive the propagated effect of design from node 6 and its upstream nodes. Node 5 and 6, on the other hand, are at the same time dominators, as they can propagate changes to others, and subordinators, as they receive the changing effects from others as well. Thus, it is not enough just to categorize the design elements into design dominator and subordinator. A quantitative approach would be of much merit.

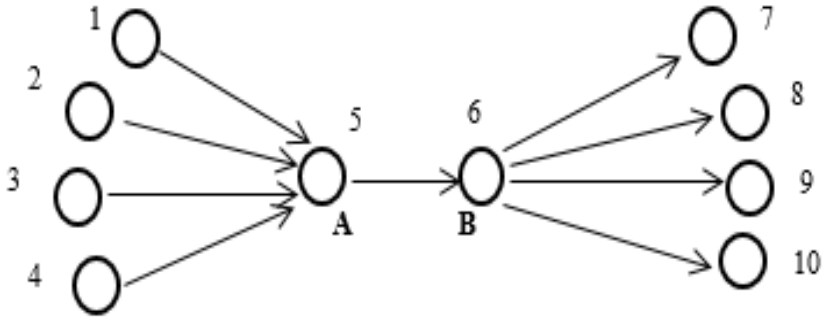


Figure 4 An example of illustrating design dominator and subordinator

The proposed method is based on graph theory. The nodes in the graph could represent design elements like subassemblies, components of the product, or features in the part of the product, depending on the level of granularity of interest. In the case study of the current chapter, for simplicity, nodes represent components or subassemblies.

Two indices are defined to depict the extent to which a design element dominates and subordinates in the whole system from design dependency analysis, namely design domination weight and subordination weight. These two indices measure the extents to which a design element is influencing, or being influenced by, other design elements. Each design element is assigned with these two indices. It aligns with the statement that a design element could be a design dominator and subordinator at the same time. The question remains is how to assign the weight values to each design element to indicate their extents. One possibility is to consider the in-degree and out-degree of each design element. However, this method is not adopted due to its localness. The following sections discuss a global method proposed in this chapter.

3.3.3 Iterative approach

This subsection presents an iterative approach to calculate the design domination and

subordination weight. A set of n nodes in a directed graph representing design elements would be considered here. Given the set of nodes, labeled 1,2,3, ...n, the links among them could be encoded in the adjacency matrix A as follows: the entry in the i^{th} row and j^{th} column of A, denoted as A_{ij} is equal to 1 if there is a dependency from node j on i, namely changes in node i would propagate to node j or design decision of node i exerts influence over design decision of node j. It is equal to 0 otherwise. The diagonal elements of A_{ij} are set to be 0. Design domination weights and subordination weights are the lists of numbers with each one associated with each of the n nodes of the network and could be represented as vectors with a dimension of $n \times 1$. Given the adjacency matrix, we start by assuming vector $\boldsymbol{t}$ and $\boldsymbol{r}$ (domination and subordination) values (randomly generated non-negative values), as is illustrated in Figure 5.

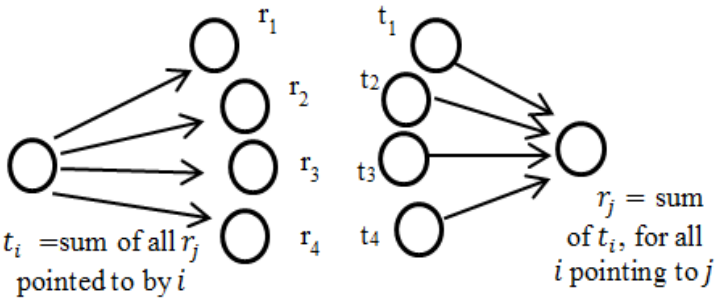


Figure 5 The basic operations adapted from Kleinberg (1999)

$$t(i) = \sum_j A_{ij} r(j) \tag{1}$$

$$r(j) = \sum_i A_{ij} t(i) \tag{2}$$

In matrix form, we have

$$t = Ar \tag{3}$$

$$r^T = t^T A \tag{4}$$

$$r = A^T t \tag{5}$$

This leads to an iterative procedure in which at each iteration $k \geq 0$ (k is the number of iteration), the design domination weights and subordination weights are updated according to:

$$t_{k+1} = (A\,A^T)t_k \tag{6}$$

$$r_{k+1} = (A^T A)r_k \tag{7}$$

Given the adjacency matrix representation A of a graph G, dimension of A is $n \times n$. A large variable $MaxIteration$. A small variable ϵ

Let $B = AA^t$

Initialize a random vector t_p with n elements, normalize t_p

Set $t = t_p$

Set $i = 0$

While $i++ < MaxIteration$

    $t = B \times t_p$

    Normalize t

    If $norm(t - t_p) < \epsilon$

        Break

    $t_p = t$

return t

Figure 6 The iterative algorithm to calculate t

For example, the iterative algorithm to calculate t is presented in Figure 6. The iterative procedure converges with normalization as k increases and the values are determined only by the structure of the network regardless of the starting estimate of design domination and

subordination values, the proof of which could be found in Kleinberg (1999) and Easley and Kleinberg (2010). Using eigenvector decomposition, one can show that this iteration with normalization converges to a limit point related to the principal eigenvectors of the matrix $A^T A$ and AA^T . The principal eigenvector is the eigenvector corresponding to the principal eigenvalue, which is the largest absolute eigenvalue. The above results are valid only when the principal eigenvalue is unique, which, as suggested in Easley and Kleinberg (2010), is often the case with real and sufficient network structure.

Note that if the design dependency graph contains a large fraction of nodes that have either only in-degree or out-degree, direct use of adjacency matrix might not lead to a good result because the flows of information stop moving when they come to the dead-end nodes, which blocks the iteration process. In this scenario, a minor adjustment is needed. The matrix A in equation 1 to 7 should be updated as:

$$A = A + A \times A + A \times A \times A + \cdots \tag{8}$$

Namely, if needed the indirect effect could also be taken into the iterative process. Note that the domination weights and subordination weights are reinforcing each other. With that being said, the subordination weight of each node is not only determined by how many nodes are influencing the receptors but also affected by how influential those nodes are. So is the domination weight - it takes into consideration both the number of nodes it could influence and the degree of the subordination weights of those nodes. For example, the calculated weight values for the graph in Figure 4 are:

$$t = (0.44,0.44,0.44,0.44,0.38,0.31,0,0,0,0) \tag{9}$$

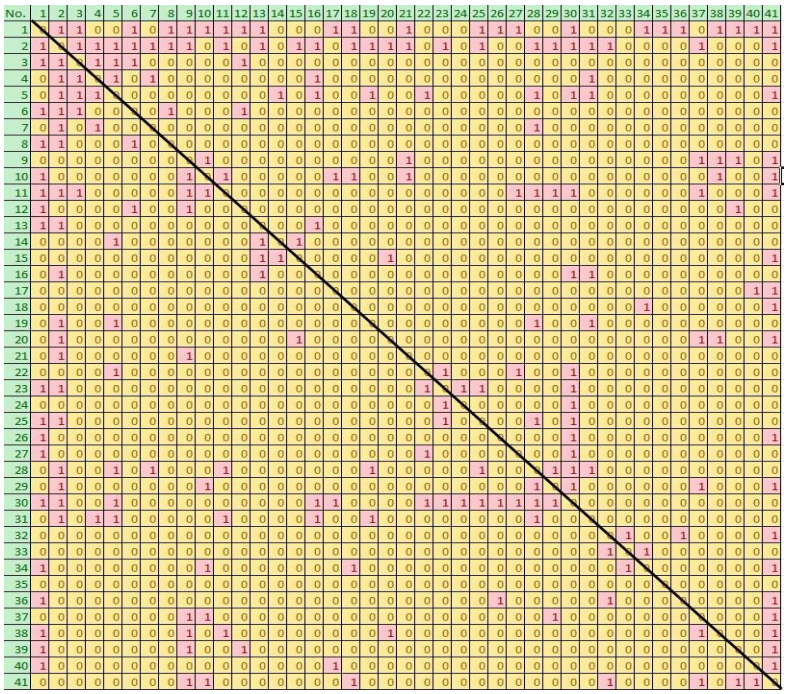
$$r = (0,0,0,0,0.31,0.38,0.44,0.44,0.44,0.44) \quad (10)$$

3.4 Case study

3.4.1 Model construction

The proposed design dependency assessment method is based on the fundamentals of DSM. The current chapter is not meant to construct the matrix, but to make use of the constructed dependency matrix and apply the proposed approach here. The model of the case study is based on Perkins's diesel engine (Hamraz et al. 2012), as is shown in Figure 7(a). The DSM representation of the diesel engine is shown in Figure 7(b). The off-diagonal elements of DSM denote the dependencies of one design element on the other, namely the design of one element influence the design of the other.

(a)



(b)

Figure 7 (a) Perkins's diesel engine (Hamraz, Caldwell, and John Clarkson 2012); (b) The adapted DSM representation of the engine

No.	Component	Design Dominanation Weight	Design Subordination Weight
1	Cylinder Head Assembly	0.383	0.321
2	Cylinder Block Assembly	0.471	0.351
3	Piton Rings Gudgeon Pin	0.139	0.190
4	Conn Rod	0.129	0.126
5	Crankshaft Main Bearings	0.238	0.181
6	Valve Train	0.123	0.136
7	Cam Shaft	0.081	0.092
8	Push Rods	0.094	0.113
9	High Pressure Fuel Pipes	0.111	0.226
10	ElectriControl Module	0.178	0.148
11	Fuel Pump	0.292	0.178
12	Fuel Injection Assembly	0.088	0.088
13	Adapter Plate/Flywheel Housing	0.096	0.124
14	Flywheel Ring Gear	0.044	0.035
15	Starter Motor	0.067	0.073
16	Sump	0.105	0.154
17	Oil Filler	0.047	0.103
18	Engine Breather	0.046	0.145
19	Oil Pump	0.104	0.122
20	Oil Filler2	0.119	0.080
21	Oil Cooler	0.067	0.133
22	Crank Pulley Damper Belt	0.081	0.082
23	Fan Drive	0.145	0.115
24	Fan Extension	0.046	0.045
25	Coolant Pump	0.150	0.166
26	Alternator Bracket	0.110	0.084
27	Belt Driven Auxiliary (Hydraulic Pump)	0.080	0.116
28	Gear Train	0.194	0.221
29	Gear Driven Auxiliary (Compressor)	0.175	0.151
30	Timing Case	0.242	0.286
31	Balancer	0.154	0.144
32	Turbocharger	0.048	0.080
33	Aircharge Cooler	0.015	0.019
34	Air Intage	0.113	0.051
35	Air Filter	0.000	0.044
36	ExhaustManifold	0.096	0.050
37	Low Pressure Fuel System	0.100	0.165
38	Fuel Filter	0.152	0.092
39	Starting Aid	0.113	0.080
40	Lifting Eyes	0.089	0.062
41	Wiring Harness	0.105	0.342

Figure 8 Design domination and subordination weights of the diesel engine case study

Note that in their original paper the DSM is constructed such that columns indicate components that initiate changes and rows receive changes, a way of demonstrating dominance

of one element over the other. In the current research, the order is reversed. Either way works if consistency is well maintained.

Rank	No.	Component	Design Domination Weight
1	2	Cylinder Block Assembly	0.471
2	1	Cylinder Head Assembly	0.383
3	11	Fuel Pump	0.292
4	30	Timing Case	0.242
5	5	Crankshaft Main Bearings	0.238
6	28	Gear Train	0.194
7	10	ElectriControl Module	0.178
8	29	Gear Driven Auxiliary (Compressor)	0.175
9	31	Balancer	0.154
10	38	Fuel Filter	0.152
11	25	Coolant Pump	0.150
12	23	Fan Drive	0.145
13	3	Piton Rings Gudgeon Pin	0.139
14	4	Conn Rod	0.129
15	6	Valve Train	0.123
16	20	Oil Filler2	0.119
17	39	Starting Aid	0.113
18	34	Air Intage	0.113
19	9	High Pressure Fuel Pipes	0.111
20	26	Alternator Bracket	0.110
21	41	Wiring Harness	0.105
22	16	Sump	0.105
23	19	Oil Pump	0.104
24	37	Low Pressure Fuel System	0.100
25	13	Adapter Plate/Flywheel Housing	0.096
26	36	ExhaustManifold	0.096
27	8	Push Rods	0.094
28	40	Lifting Eyes	0.089
29	12	Fuel Injection Assembly	0.088
30	7	Cam Shaft	0.081
31	22	Crank Pulley Damper Belt	0.081
32	27	Belt Driven Auxiliary (Hydraulic Pump)	0.080
33	15	Starter Motor	0.067
34	21	Oil Cooler	0.067
35	32	Turbocharger	0.048
36	17	Oil Filler	0.047
37	24	Fan Extension	0.046
38	18	Engine Breather	0.046
39	14	Flywheel Ring Gear	0.044
40	33	Aircharge Cooler	0.015
41	35	Air Filter	0.000

Rank	No.	Component	Design Subordination Weight
1	2	Cylinder Block Assembly	0.351
2	41	Wiring Harness	0.342
3	1	Cylinder Head Assembly	0.321
4	30	Timing Case	0.286
5	9	High Pressure Fuel Pipes	0.226
6	28	Gear Train	0.221
7	3	Piton Rings Gudgeon Pin	0.190
8	5	Crankshaft Main Bearings	0.181
9	11	Fuel Pump	0.178
10	25	Coolant Pump	0.166
11	37	Low Pressure Fuel System	0.165
12	16	Sump	0.154
13	29	Gear Driven Auxiliary (Compressor)	0.151
14	10	ElectriControl Module	0.148
15	18	Engine Breather	0.145
16	31	Balancer	0.144
17	6	Valve Train	0.136
18	21	Oil Cooler	0.133
19	4	Conn Rod	0.126
20	13	Adapter Plate/Flywheel Housing	0.124
21	19	Oil Pump	0.122
22	27	Belt Driven Auxiliary (Hydraulic Pump)	0.116
23	23	Fan Drive	0.115
24	8	Push Rods	0.113
25	17	Oil Filler	0.103
26	7	Cam Shaft	0.092
27	38	Fuel Filter	0.092
28	12	Fuel Injection Assembly	0.088
29	26	Alternator Bracket	0.084
30	22	Crank Pulley Damper Belt	0.082
31	20	Oil Filler2	0.080
32	39	Starting Aid	0.080
33	32	Turbocharger	0.080
34	15	Starter Motor	0.073
35	40	Lifting Eyes	0.062
36	34	Air Intage	0.051
37	36	ExhaustManifold	0.050
38	24	Fan Extension	0.045
39	35	Air Filter	0.044
40	14	Flywheel Ring Gear	0.035
41	33	Aircharge Cooler	0.019

Figure 9 The rank of the design domination and subordination weights of the diesel engine

3.4.2 Design domination weight and subordination weights

After applying the algorithm presented in the above section, the results of design domination weights and subordination weights, as well as their rankings, could be found in Figure 8 and Figure 9. The distribution of these two sets of weights for different design elements could be found in Figure 10.

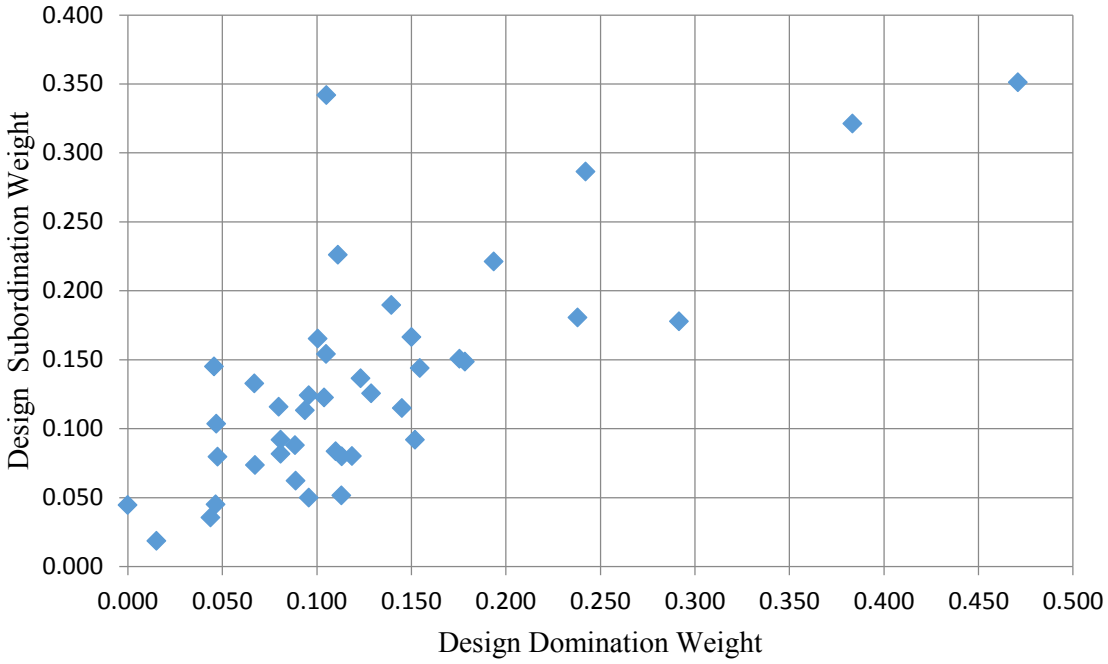


Figure 10 Design domination and subordination weights distribution for the diesel engine

3.4.3 Discussion

The design dependency matrix shows the causal relationships among components or subassemblies of the diesel engine. The design domination weights of the engine components and the corresponding ranks help to identify those influential components regarding design dependencies. The design subordination weights of the engine present the degree to which each component is subject to the design decisions or changes of other components.

Different design strategies could be formed from the insight of design dependencies assessment of design elements. For example, for the design elements with high design domination weights and low subordination weights, the design parameters, if possible, should be frozen earlier than later. The design decision of such design elements is expected to have a massive impact on the system. For the design elements with high subordination weights and low domination weights, it makes more sense to fix the design parameters later after the design parameters of those "influential" design elements have been determined, like the "wiring harness" in Figure 9. The changes of such design elements are not expected to cause much knock-on effects. For the design elements with both high domination weights and subordination weights, e.g., cylinder block assembly, their design is expected to require more and iterative efforts to carry out as their behavior could hardly be fully determined. Hence it deserves more attention from the designers. Least risk and redesign efforts are found in those design elements with both low design domination and subordination weights, e.g., aircharge cooler. Such insight helps to plan the design process of a new product or the redesign of existing products.

In addition, the design efforts of different design elements could be prioritized to assist in allocating design resources in terms of the risk involved in, for example, the change propagation effects. The calculated weights are related to the position of the corresponding design elements in the product dependency network independent of the "real content" within individual design element, which renders the method generically applicable to the design dependency analysis of other artifacts. It highly depends on the quality of the dependency network.

The calculated weights can be used to assess the impact of the design changes to certain design elements. The impact here is defined as the scope of which certain design change is capable to propagate to. For example, if a design change is to be made to a design element with

high domination weight, it is likely that such design change will have a bigger impact as it is likely to propagate changes to many other design elements.

Current research does not restrict the category of dependencies used to construct the dependency network. It is expected that proposed method, due to its generic nature, is well applicable in analyzing geometrical, structural, functional, and assembly or manufacturing dependencies. For example, in the combination of geometrical, structural, and functional design dependencies of the design element, designers can have a more thorough picture of the intricate inter-relations among the design element.

However, there are limitations. It is found that the interactions among design elements are implicit in this chapter. Although the proposed approach can quantify the role and position of design elements in the design dependency graph, the result is as good as the quality of the design dependency graph. From the analysis results of the dependency graph borrowed from literature (Hamraz, Caldwell, and John Clarkson 2012), it is safe to say that they are mostly mixed dependencies, i.e., they contain both functional and structural dependencies. For example, the cylinder block assembly is structurally important because most other parts are assembled onto it. It is functionally important because it is where the conversion of the heat energy to mechanical energy happens. Both types of dependencies are important in the product design process.

3.5 Chapter summary

This Chapter introduces a method for assessing the complexity of design elements and analyzing design dependencies. Two indices, design domination weight and design subordination weight, are proposed to assess such behaviors, based on which designers could better evaluate the scope and impact of design changes and make better decisions of allocating resources and scheduling

design activity, e.g., the recommended design strategies according to different behaviors due to design dependency. This method could be integrated into current PLM software to offer the functionality of design dependency assessment and change impact assessment. Most importantly, the result of the case study also shows that the design dependencies are of a mixed type, including structural and functional dependencies.

4 Feature Dependencies Assessment in CAD Models

4.1 Chapter introduction

Feature-based CAD models are typically created with a combination and series of operations with predefined features, e.g., block feature, hole feature, blend feature, and extrude feature. In most systems, sketch and *Boolean* operations are also frequently used as feature operations. CAD model reuse boosts product development process (Camba et al. 2016). Unfortunately, not all CAD models are reusable. To reach a more robust CAD modeling strategy, a better understanding of the nature of the CAD model construction is necessary, i.e., modeling intents.

Modeling intents are different from design intents. Design intents are what associate functions with product structures, i.e., the reasons why a product has specific structures. Design intents convey functional design considerations to product structures. Modeling intents are restricted to the intents and rationales behind the CAD model construction. There are two levels of modeling intents, i.e., the reasons why models are constructed in certain ways to, firstly, conform to the physical structures, and secondly, comply with functional design considerations.

Modeling intents are not explicitly available in the model. Users, with the same set of feature operations at hand, might construct visually identical product geometry with different modeling procedures, which result in different feature dependencies (Bidarra and Bronsvoort 2000). For some examples of feature dependencies in CAD, see Figure 11. So, it is not enough to analyze the shape information. To reveal modeling intents in CAD models, the analysis of applied features is a way to go. More specifically, the analysis of feature dependencies is necessary to unveil modeling intents.

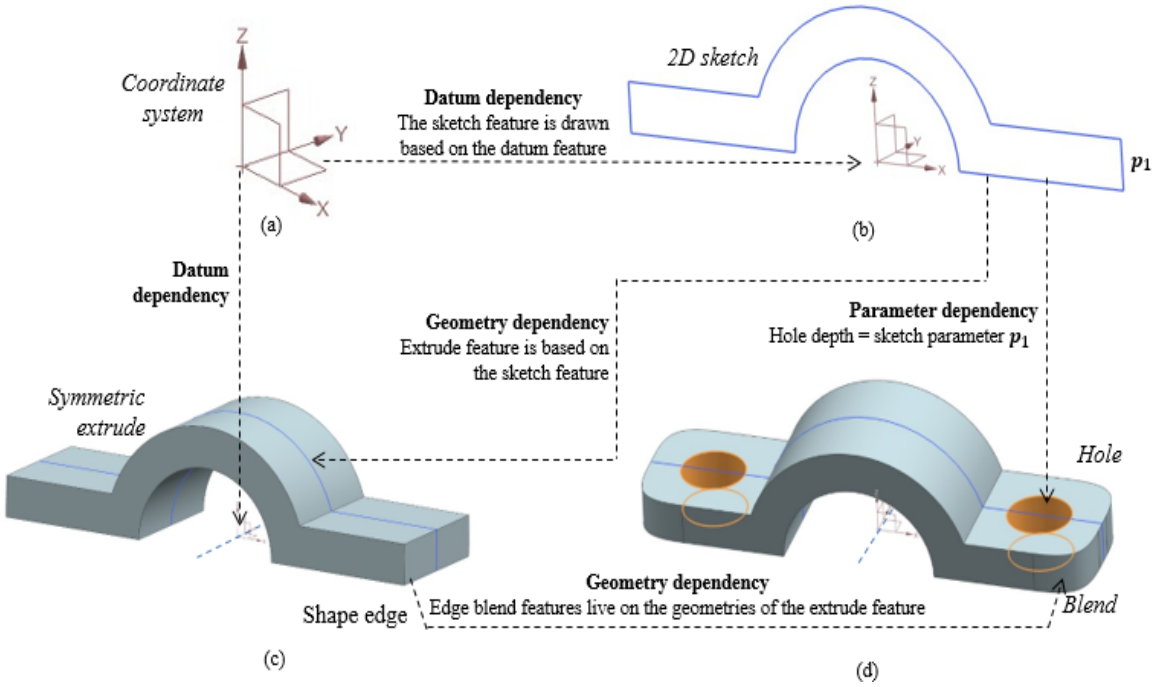


Figure 11 Some examples of feature dependencies in CAD

The rest of the chapter is organized as follows. Section 4.2 presents a few graph centrality metrics to be used in the current chapter. Section 4.3 shows the observed properties of the feature dependency graph that could be made use of, as well as the algorithm to extract feature information from CAD models to construct the feature dependency graph. Section 4.4 presents the implementation procedure. A few examples are studied in Section 4.5. The last section concludes the chapter.

4.2 Centrality metrics

Centralities of nodes are critical measures for a graph. Some are local measures, e.g., degree centrality, some are global in the sense they measure the centrality of the specific node relative to the rest of the network, e.g., closeness, betweenness, and eigenvector centrality. Degree centrality measures how many edges are connected to each node in the graph. In a direct graph, degree centrality could be further categorized into in-degree and out-degree centrality. The larger

in or out connecting edge number is, the bigger the in or out degree value is. The value could be normalized by dividing by the maximum possible number of the connections. Betweenness centrality is a measure to quantify the number of times a node acts as a bridge along the shortest path between two other nodes, the value of which can also be normalized. Closeness centrality of a node in the graph is the reciprocal of the length of the total shortest path between the node and all other nodes in the graph. Eigenvector centrality is interesting because it says that the centrality of a node depends on its neighbors' centrality, and its neighbors also depend on their neighbors' centralities. The values of interests are contained in the eigenvector corresponding to the largest eigenvalue. The approach introduced in Chapter 3 is a variation of eigenvector centrality.

The formulas for calculating the above-mentioned centralities are listed in the Table 3, where A denotes the adjacency matrix of the graph. Another variation of eigenvector centrality, applied in link analysis using hubs and authorities in information networks and World Wide Web (Easley and Kleinberg 2010) and in determining design domination weight and design subordination weight in dependency analysis of design elements in product development (Cheng and Ma 2014) is to calculate the dominant eigenvector of, instead of the adjacency matrix of the graph, the multiplication of adjacency matrix with its transpose.

The key difference between degree centrality and closeness centrality is that degree centrality is based on the number of connections a node has whereas closeness is measured based on the length of the shortest path to all other nodes. The major difference between degree centrality and eigenvector centrality is that degree centrality is local whereas eigenvector centrality is global because it considers both the direct and indirect connections. A node with larger degree centrality does not necessarily have higher eigenvector centrality because its

immediate neighbors might have low or null eigenvector centrality. Similarly, a node with significant eigenvector centrality is not guaranteed to have high degree centrality because the nodes it connects to might be small in number but large in value.

Table 3 Formulas to calculate centralities (Katz 1953; Freeman 1977; Bonacich 1987; Bonacich and Lloyd 2001)

Degree centrality	Non-normalized	$C_D(i) = \deg(i)$
	Normalized	$C'_D(i) = \frac{C_D(i)}{N-1}$
Betweenness centrality	Non-normalized	$C_B(i) = \left[\sum_{j,k} g_{jk}(i)/g_{jk}\right]$
	Normalized	$C'_B(i) = \frac{C_B(i)}{(N-1)(N-2)}$
Closeness centrality	Non-normalized	$C_i(i) = \left[\sum_{j=1}^{N} d(i,j)\right]^{-1}$
	Normalized	$C'_i(i) = \frac{C_i(i)}{N-1}$
Eigenvector centrality and its variations	Eigenvector	$\lambda \boldsymbol{e} = \boldsymbol{A}\boldsymbol{e}$
	Katz	$C_{Katz} = ((\boldsymbol{I} - \alpha \boldsymbol{A}^{\boldsymbol{T}})^{-1} - \boldsymbol{I})\boldsymbol{1}$
	Alpha	$C_{Alpha} = (\boldsymbol{I} - \alpha \boldsymbol{A}^{\boldsymbol{T}})^{-1}\boldsymbol{e}$
	Bonachich Power	$c(\alpha, \beta) = \alpha(\boldsymbol{I} - \beta \boldsymbol{A})^{-1}\boldsymbol{A}\boldsymbol{1}$

Other metrics are also available to measure certain properties of a graph, for example, graph density, which is a fraction number that can be used to describe the extent of connections in the graph. Since current chapter focuses on centralities measurements of feature dependency graph and its implication in unveiling modeling intents, investigation of other metrics is out of the scope of this chapter.

4.3 Feature dependency graph

4.3.1 Characteristics of feature dependencies graph

The graphs constructed in this research are based on feature dependencies (Bidarra and Bronsvoort 2000). Certain characteristics of the feature dependency graph could be observed. Feature dependencies are *non-reflexive*, i.e., a feature cannot depend on itself, feature dependencies are *nonsymmetrical*, i.e., two features cannot mutually depend on each other, and feature dependencies are *transitive*, i.e., if feature a depends on feature b and feature b depends on feature c, then feature a also depends on feature c (Li et al., 2009). Mathematically, the above-mentioned characteristics could be expressed as:

Property 1: Non-reflexive

$$\forall\, a \in V, \neg(a\boldsymbol{R}a)$$

Property 2: Nonsymmetrical

$$\forall\, a, b \in V, a\boldsymbol{R}b \neq\ b\boldsymbol{R}a$$

Property 3: Transitive

$$\forall\, a, b, c \in V\colon (a\boldsymbol{R}b) \wedge (b\boldsymbol{R}c) \Rightarrow (a\boldsymbol{R}c)$$

where $\boldsymbol{R}$ denotes dependency relations, ¬ is negate.

With these characteristics, it can be deduced that the feature dependency graph is directed and acyclic, which is called Acyclic and Directed Feature Dependency Graph (ADFDG) in this research, where the nodes, or vertices V, are the features and edges E depict the feature dependencies. Therefore, an ADFDG G is defined as:

$$G = (V, E)$$

4.3.2 Construction of ADFDG

Figure 12 shows the algorithm to construct ADGFG from any feature-based CAD part model. The algorithm to find features in the model and their relations is straightforward. It makes use of two key points. First, the part model contains every feature used to construct the model. Secondly, each feature has pointers to its children features. Representation wise, one could use either edge list, adjacency matrix, or adjacency list for the graph. Note that a map data structure is like a dictionary, which contains different pairs or entries, the first element of which is key and second values. Thus, this algorithm gives an adjacency list representation for the ADGFG.

Algorithm to create adjacency list representation of ADFDG

Initialization: Given a feature-based CAD part p, set $V = \{\emptyset\}$, map $A = \{\emptyset\}$

popularize set V with all features in part p

for each element v_i in feature set V for part p **do**

create an empty list of features $L_i = [\emptyset]$

for each cf_j belonging to the immediate children features of v_i **do**

insert cf_j into list L_i

insert into map A with pair v_i and L_i

return the map A

Figure 12 The algorithm to extract feature information

With the above-mentioned information, it becomes obvious how to generate the feature dependency graph. The first step is to get all the features in the part model (to form a set of nodes in the graph), then for each of the feature get its immediate children features to construct the adjacency list. Note Li et al. (2009) suggested removing the non-shape-related features, e.g., datum features and sketches, by suppressing the vertices and removing relevant edges because they are not solids with volumes. However, those non-solid features are critical in the process of CAD modeling and have great influence on the model construction because the changes of, for example, references features, would generally lead to the update of the model. Thus, in the current research if an entity is treated as a feature in the CAD system it is included in the generation of ADFDG.

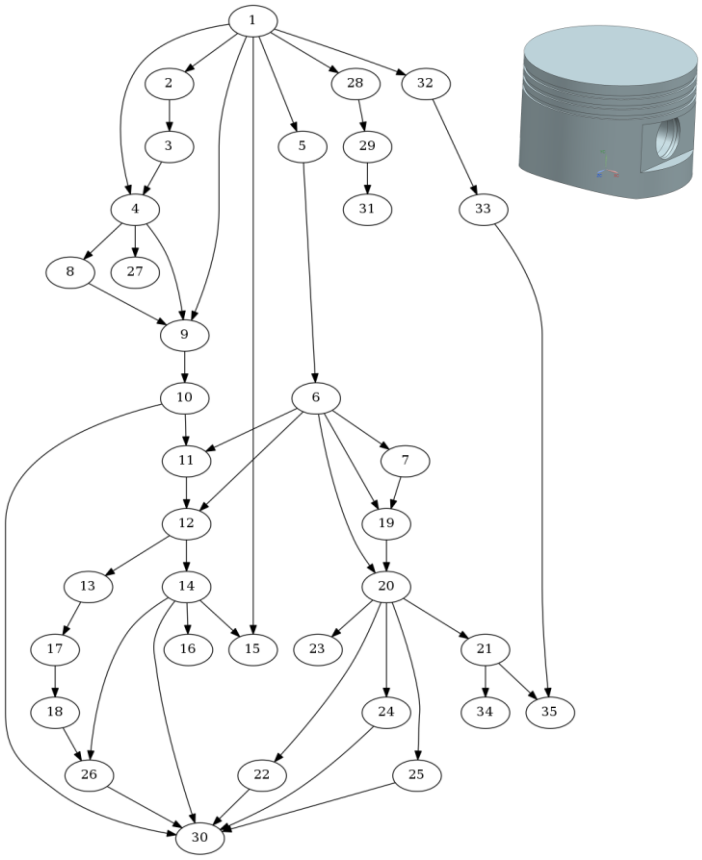


Figure 13 Generated feature dependency graph of a piston example

With the information of each feature and its children features in the part model, i.e., its adjacency list representation, its ADFDG can be virtualized. An example of the feature dependency graph for a piston used in reciprocating engines is illustrated in Figure 13.

4.4 Implementation procedure

The general framework of the current research is presented in Figure 14. It starts with a constructed feature-based CAD model with all the model history and feature information. Then feature information is extracted from the model with API programming to construct the ADFDG based on the algorithm introduced in Figure 12. With the ADFDG at hand, visualization and centrality analysis of the graph could be carried out.

Implementation-wise, ADFDG in the current research is created from models constructed with Siemens NX and retrieved via its Open C++ Application Programming Interfaces (APIs). Note that all the objects in the model creation, including features, have tags to identify them uniquely in the current session. Current research generates the feature dependency graph with the feature tags and then converts them into unique integers starting from 1. The numbering indicates which feature operation comes earlier and which comes later, i.e., the larger number means the feature is constructed later in the process. Each node has the integer id plus the information of feature type.

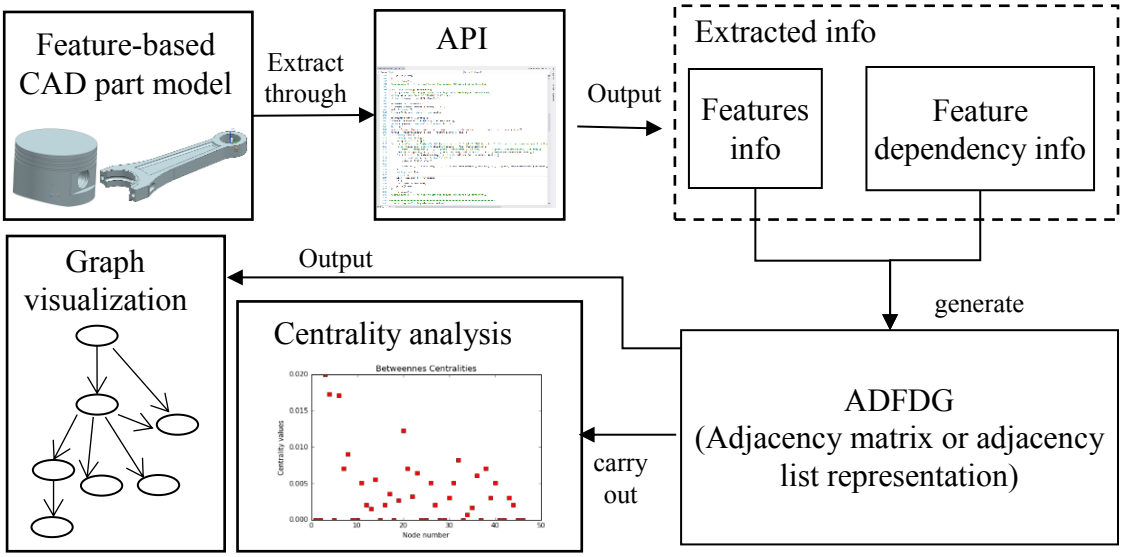


Figure 14 The general implementation procedure for extracting and analyzing ADFDG

Note that all feature operations are recorded in the system and most of them are accessible to end users from the CAD system, e.g., the part navigator in NX. The reason that not all the features contributing to the model construction are presented in part navigator is that some feature operations are implicit, i.e., created by the CAD system automatically to assist the modeling, instead of explicitly created by the user. Those implicit features, although not accessible to end users from the Graphical User Interface (GUI), could be extracted with API

programming. Figure 15 shows that the algorithm to extract feature information has been implemented as an add-on to the NX. If a user pushes the button, the system will call the custom-made and compiled *DLL* file as shown in the figure. The result of the execution is a text file containing feature and feature dependency information to be further processed. The code for feature finder and graph visualization can be found in the Appendix 1.

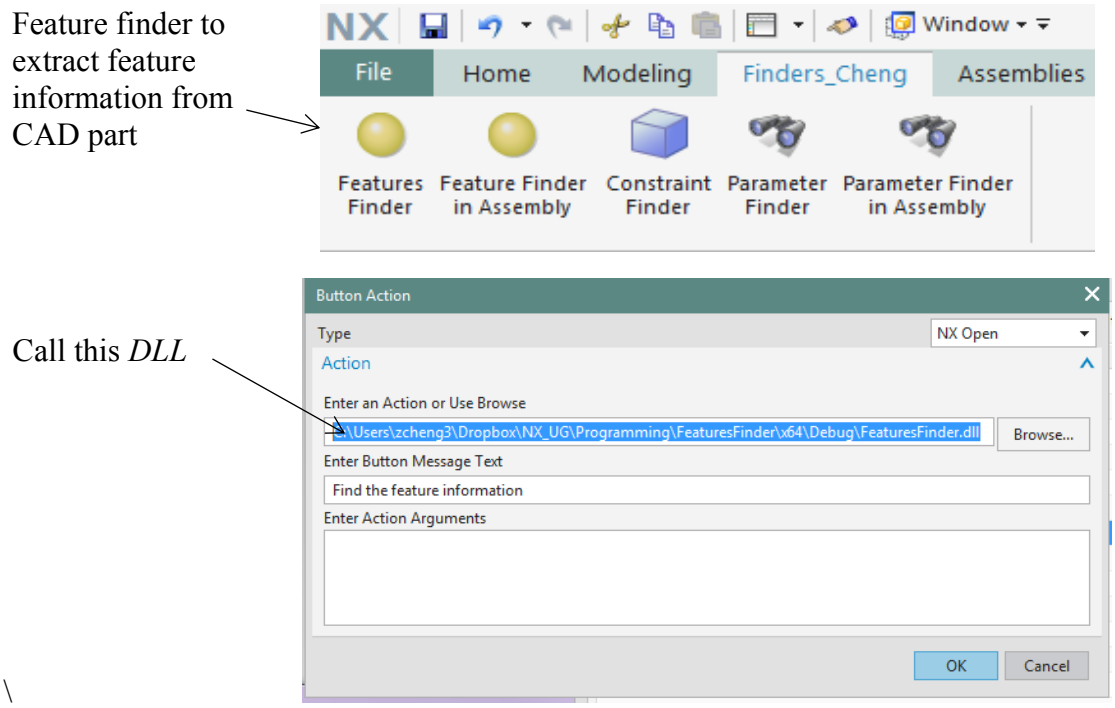


Figure 15 The implemented feature finder with GUI in the NX

4.5 Case studies for visualization and analysis of ADFDG

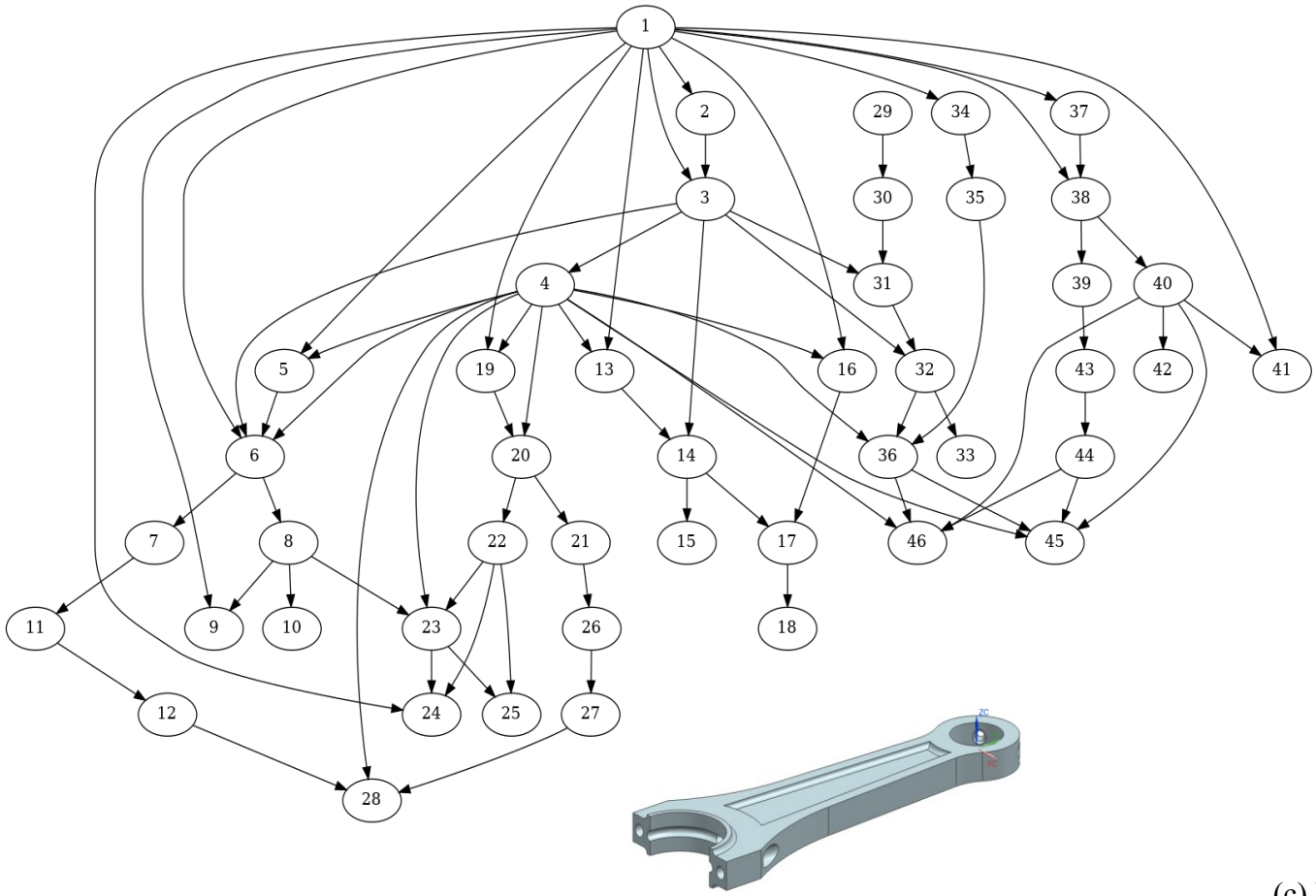
This section provides three case studies to prove that the proposed procedure is well applicable to extract, visualize, and analyze feature-based CAD models. It would be seen that each of these three parts has its own characteristics. The first model is a commonly seen, standardized solid part; the second model contains freeform geometries; the last one is basically a shell structure.

4.5.1 A Connection rod example

A connection rod model in an inner combustion engine design is taken as the first example to illustrate our proposed method. The connection rod is a component used in the reciprocating engine. This component comprises dozens of feature operations to construct. It is believed to be typical to show the effectiveness of the proposed method and yet not too complicated to make it hard for the readers to follow.

The connection rod CAD model, shown in Figure 16 (b), is constructed with feature operations in a commercial CAD system (Siemens NX) by the author, the model history and ADFDG of which is shown in Figure 16 (a), together with part of the index to feature type map in (c). Figure 17 provides the adjacency list and edge list representation of the resulting feature dependencies. The resulting centrality values for different features of connection rod ADFDG with feature names could be found in the Appendix 2.

Figure 18 shows two prominent features in the connection rod example, which is the result of the different centrality analyses provided in Figure 19. Figure 20 gives correlations of different centralities, where some key features are numbered and their correlation values are given in c. It is found that the connection rod model has a few dominant features.



(a) ADFDG visualization

(b) CAD model

Index	Feature type
1	Datum_Csys
2	Datum_Csys
3	Sketch
4	Extrude
5	Datum_Csys
6	Sketch
7	Extract_String
8	Extrude
9	Mirror Feature
10	Instance Feature
11	Extract_String
12	Extrude
13	Datum_Csys
14	Sketch
15	Extrude

(c) Part of the index to feature type map

Figure 16 A Connection rod case study

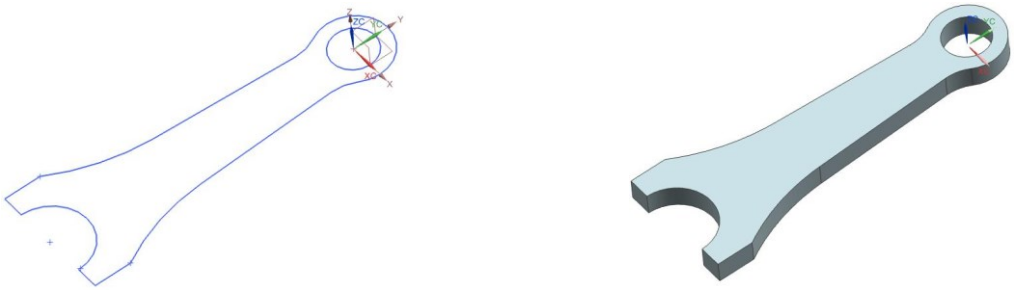
Adjacency list representation: {1: [2, 3, 5, 6, 9, 13, 16, 19, 24, 34, 37, 38, 41], 2: [3], 3: [4, 6, 14, 31, 32], 4: [5, 6, 13, 16, 19, 20, 23, 28, 36, 45, 46], 5: [6], 6: [7, 8], 7: [11], 8: [9, 10, 23], 9: [], 10: [], 11: [12], 12: [28], 13: [14], 14: [15, 17], 15: [], 16: [17], 17: [18], 18: [], 19: [20], 20: [21, 22], 21: [26], 22: [23, 24, 25], 23: [24, 25], 24: [], 25: [], 26: [27], 27: [28], 28: [], 29: [30], 30: [31], 31: [32], 32: [33, 36], 33: [], 34: [35], 35: [36], 36: [45, 46], 37: [38], 38: [39, 40], 39: [43], 40: [41, 42, 45, 46], 41: [], 42: [], 43: [44], 44: [45, 46], 45: [], 46: []}

(a)

Edge list representation: [(1, 2), (1, 3), (1, 5), (1, 6), (1, 9), (1, 13), (1, 16), (1, 34), (1, 19), (1, 41), (1, 24), (1, 38), (1, 37), (2, 3), (3, 32), (3, 4), (3, 14), (3, 6), (3, 31), (4, 16), (4, 19), (4, 20), (4, 5), (4, 6), (4, 23), (4, 36), (4, 28), (4, 13), (4, 46), (4, 45), (5, 6), (6, 8), (6, 7), (7, 11), (8, 9), (8, 10), (8, 23), (11, 12), (12, 28), (13, 14), (14, 17), (14, 15), (16, 17), (17, 18), (19, 20), (20, 21), (20, 22), (21, 26), (22, 24), (22, 25), (22, 23), (23, 24), (23, 25), (26, 27), (27, 28), (29, 30), (30, 31), (31, 32), (32, 33), (32, 36), (34, 35), (35, 36), (36, 45), (36, 46), (37, 38), (38, 40), (38, 39), (39, 43), (40, 41), (40, 42), (40, 45), (40, 46), (43, 44), (44, 45), (44, 46)]

(b)

Figure 17 The adjacency list and edge list representation of the ADFDG for the connection rod CAD model



(a) At index 3 (b) At index 4

Figure 18 Two key non-datum features of the connection rod

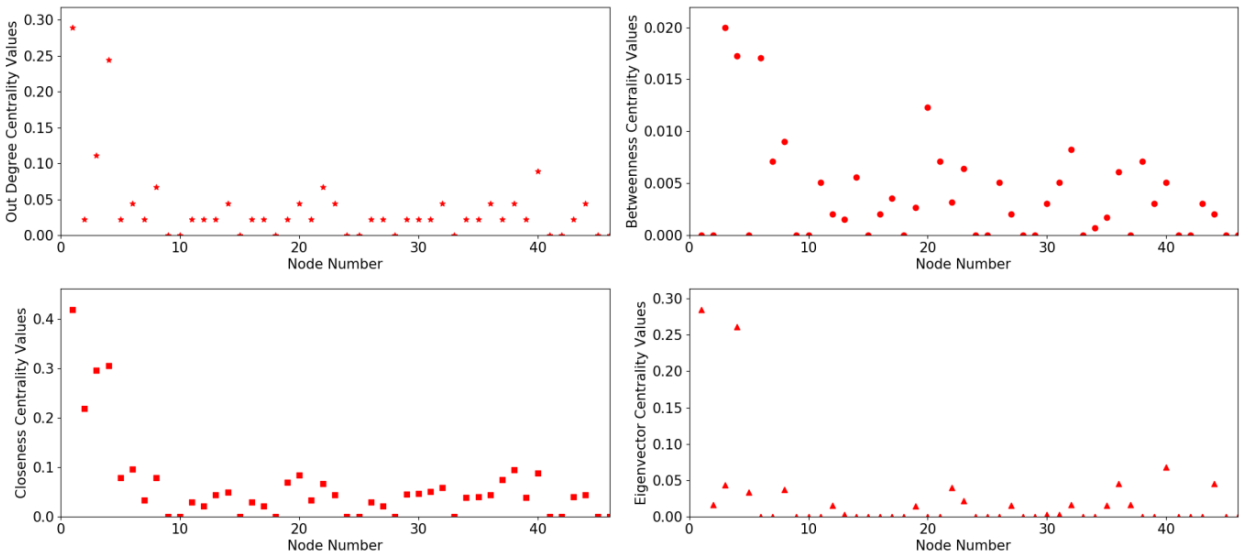


Figure 19 Centralities of the connection rod case study

Those most critical features for connection rod are the ones constructed in the beginning of the modeling process, i.e., feature 1, 3 and 4, that influence the following feature operations a lot. It could be seen that it is reasonable because for connection rod many features are built on top of the features that generate the overall shape, which could be a characteristic of the connection rod. It is predictable that for some other mechanical parts one might found more numbers of dominant shapes upon which smaller features are built. Hence, the resulting ADFDG and centrality analyses would be totally different. It could be said that on the one hand centrality analysis helps to reveal critical features in the model construction, on the other hand, helps to identify the characteristics of the model.

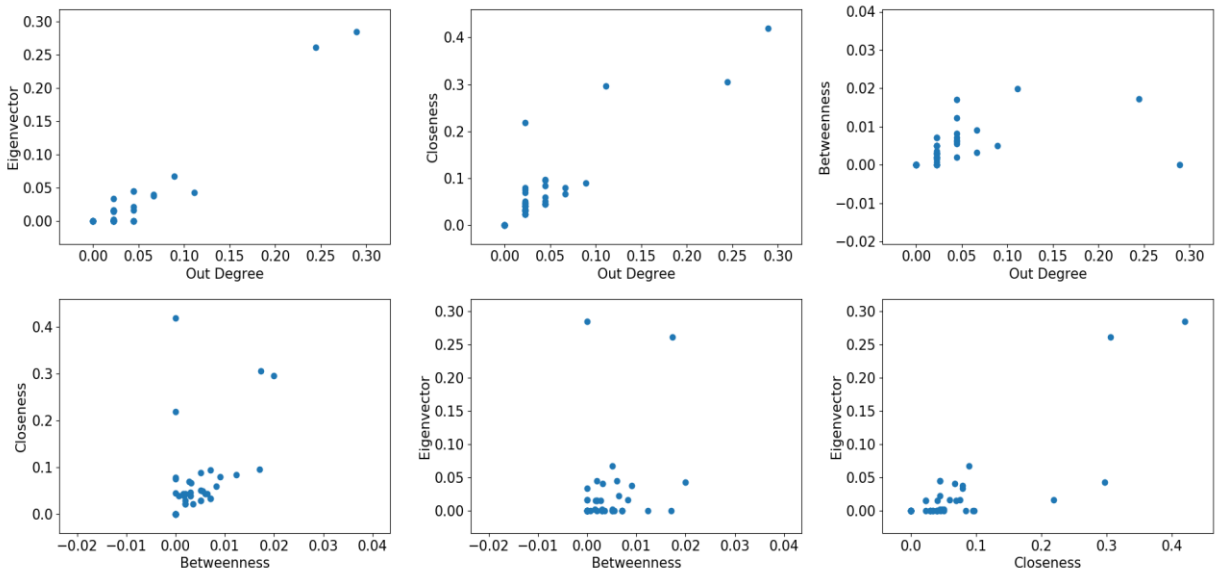


Figure 20 Correlations for the centralities of connection rod case study

4.5.2 Sports car seat and trigger switch cases

To demonstrate the generality of the proposed approach, two more cases are presented in this subsection. Figure 21 shows a sports car seat in (a) and the virtualization of its ADFDG in (b). The sports car seat model is taken from GrabCAD [3]. Some key features of the sports car model could be found in Figure 22. Figure 23 presents the centrality values for the sports car seat model. A few influencing features could be identified. Figure 24 shows a trigger switch CAD model in (a), and its corresponding ADFDG in (b). This is an example model from Siemens NX[4]. Figure 26 gives the centrality values for the trigger switch model. Different patterns of the distributions of the centrality values could be observed. Part of the map from index to feature type can be found in Appendix 2 to save space. The influencing features could be identified with the visualization and centrality analysis. It is safe to conclude that the proposed approach is generally applicable in other feature-based CAD models.

[3] Sports Car Seat, GrabCAD, https://grabcad.com/library/sports-car-seat, accessed January 2017,
[4] Siemens NX https://www.plm.automation.siemens.com/en_us/products/nx/ accessed January 2017

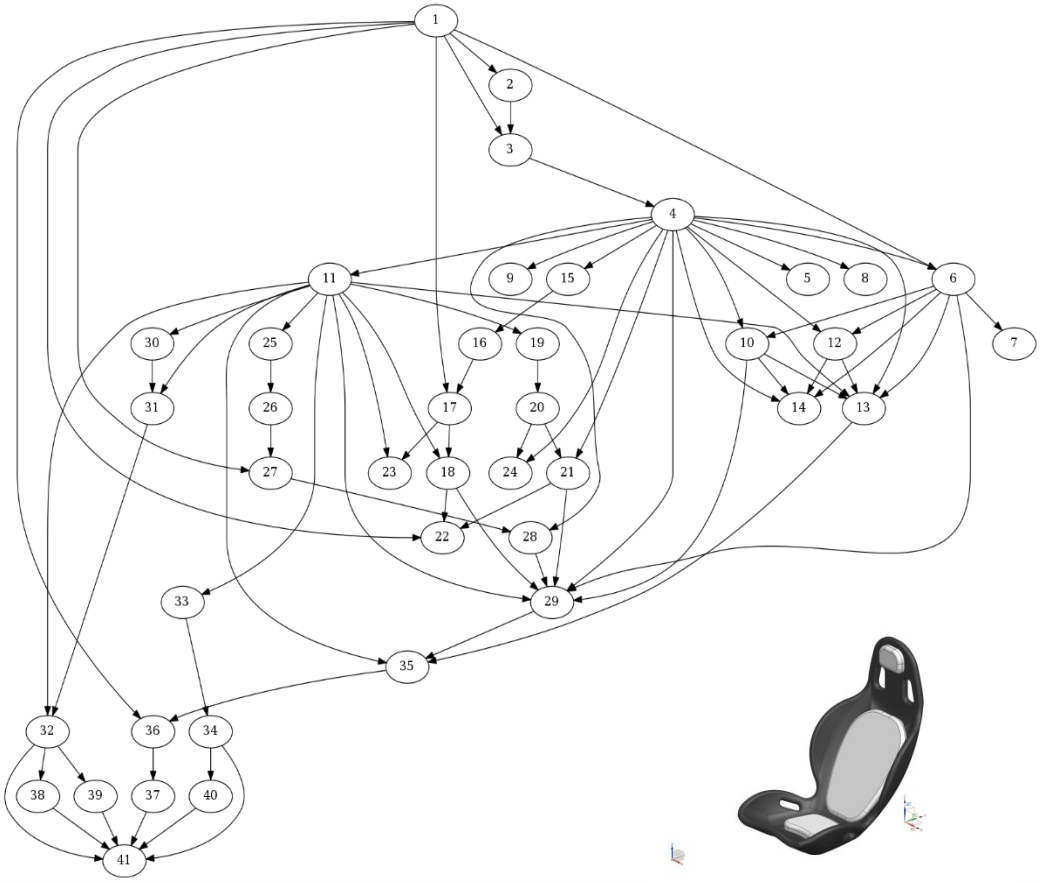


Figure 21 A sports car seat model and its ADFDG

(a) At index 3 (b) At index 4 (c) At index 11

Figure 22 Some key features of the sports car seat model

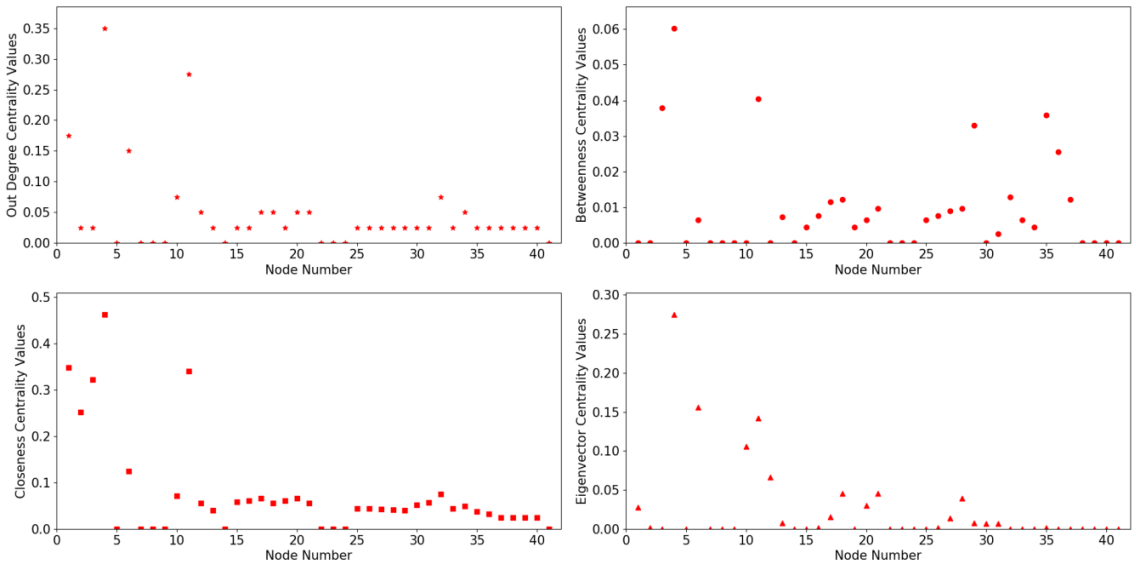


Figure 23 Centrality values for the sports car seat model

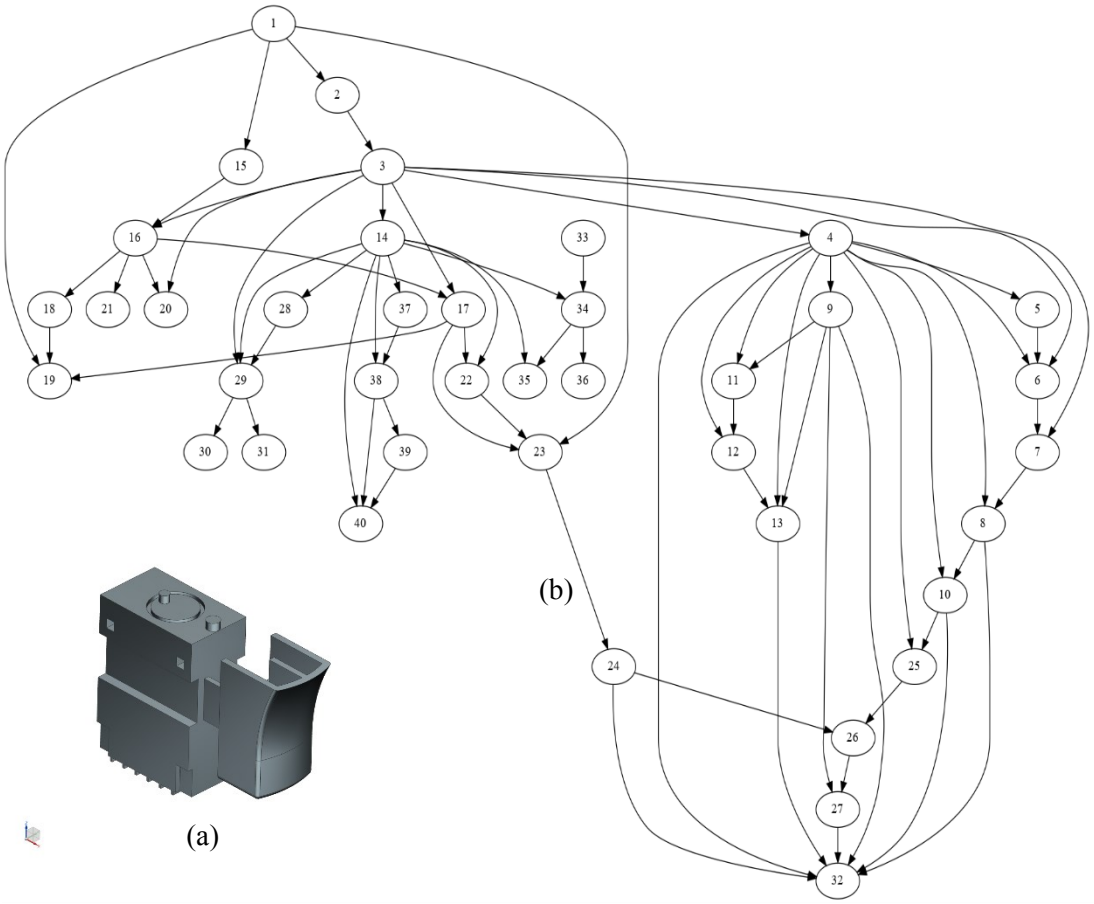
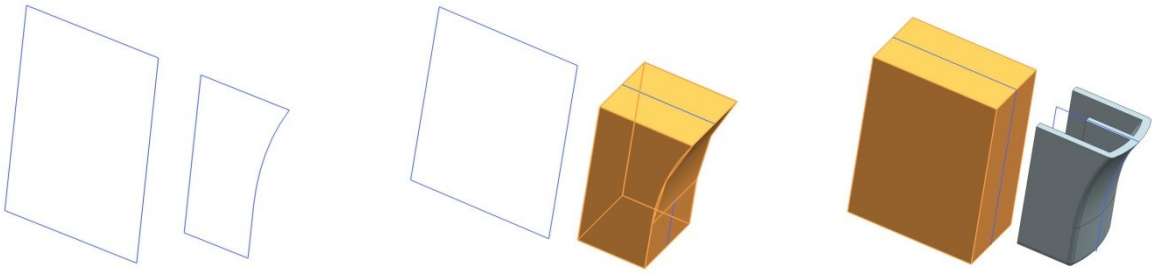


Figure 24 A trigger Switch model and its ADFDG



(a) At index 3 (b) At index4 (c) At index 14

Figure 25 Some key features of the trigger switch model

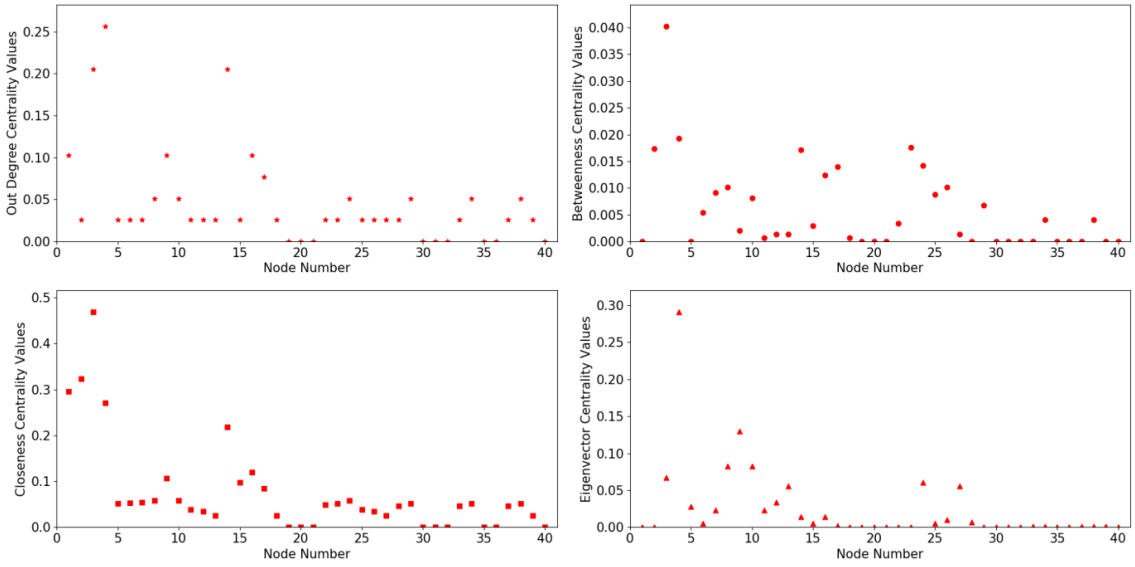


Figure 26 Centrality values for the trigger switch model

4.5.3 Discussion of the case studies

The graphical representation of the feature dependencies provided by ADFDG offers engineers a more organized view of the model construction, where interactions among feature operations are easily seen. Many graph properties are exploitable to give insights into understanding modeling intents for an individual model. Current research studies centrality properties of ADFDG, which reveals the information about which set of features are critical from the perspective of network topology.

Different metrics interpret the graph differently. They might not always agree on which features are important. Users need to choose the centrality metrics according to their application. For example, if one wants to find out which is the feature that other features directly depend on, he/she need to use the degree centrality. If one need also consider indirect dependencies, he/she could use one of the eigenvector centralities.

By looking at the identified critical features, engineers can start to ponder whether it is reasonable based on their engineering judgement, i.e., whether the applied modeling procedure is suitable or not. Ideally, the modeling intents should not only conform to the structural requirements but also comply with functional considerations of the design. So, engineers need to check whether or not the modeling intents reflected from the ADFDG comply with the structural and functional considerations of the design. They can start with the critical features. It also provides a means to review the quality of the constructed model.

Not every designer adopts a modeling methodology. Some just follow their own habits or construct models based on their experiences. There is no standard way or consensus on how to build CAD models. Visualization and analysis of the ADFDG would reveal how a model constructed by an experienced user is different from the one constructed by a new user, which provides guidance for CAD training. It is observed that experienced CAD users build models in a way that minor features are depending on major features, which could be confirmed by their resulting ADFDG. What junior CAD users can learn is how to manage the feature dependencies.

Users with different levels of experiences tend to construct the model in different fashion, which is reflected by the resulting feature tree. In terms of the interpretation of the charts when the number of features increases, there is not much difference because there are always a few

dominant features. The computational approach, given that the rules are correct, is more objective than visual observation. Potentially many exploitable engineering knowledge aspects can be revealed through this approach; that prospect warrants more future research. For example, more discoveries can be expected in the direction of merit comparison of different feature embodiment solutions.

However, it could be observed from the centrality analysis that the modeling intents are mostly geometry-centric, or form-centric for most parts. There is hardly any functional design consideration in the model construction process.

4.6 Chapter summary

This chapter proposes an intelligent knowledge discovery scheme to unveil engineering modeling intents in CAD models via centrality analysis with a type of automatically-generated feature dependency graph. An algorithm has been developed to retrieve feature dependency information from CAD models, and, instead of consulting designers or engineers to build up the network for products, to generate ADFDG for both visualization and analysis purposes. Posterior examination of the modeling intents could reveal engineering constraints applied in those CAD models. Current chapter focuses on one important aspect of the graph properties, i.e., centrality analysis. It is observed that most CAD models are constructed in a form-centric manner without embedding functional design considerations.

5 Functional Feature Framework

5.1 Chapter introduction

With the procedural part modeling, characterized by the sequential feature operations, different users might apply different modeling procedures and strategies to create a model, the effects of which are unneglectable, especially with the parametric capability in modern CAD. Moreover, different modeling approaches not only require different efforts when constructing a part for the first time, especially for those complex parts, but also result in different amount of reworks when design changes are made. The procedural modeling approach lacks the expressiveness to convey functional design considerations of design.

Engineering design is a complex process where functional design plays a central role in the conceptual design stage (Pahl et al. 2007). Recall that in the literature review, the importance of functional design has been pointed out. Chapter 3 of this thesis shows that design dependencies are usually not referred to as having only structural ones. They are implicit and of mixed type, including, for example, functional dependencies. Chapter 4 of this thesis by feature dependency analysis with CAD models revealed that CAD model construction is form-centric and is in need of an approach to embed functional design considerations. Structural considerations are important. But a complete picture should also include functional design considerations. It is needed to bridge the gap between design in general and CAD modeling activities.

The aim of the current chapter is to take an initiative in that direction with a new type of feature, functional feature. An overview of the functional feature concept will be introduced in section 5.2, where functional feature cube depicting the relations among function, structure, and

behaviors of design is presented. The semantic definition of functional feature is introduced in section 5.3, followed by behavior modeling with physics feature in section 5.4. Abstract geometry feature is a new key concept in the functional feature, which is discussed in section 5.5, followed by constraints and parameterization method in section 5.6. Section 5.7 gives a general approach to design with functional features. The last section concludes the chapter.

5.2 Overview of functional feature

The motivation of functional feature is to integrate conceptual functional design with procedural feature operations in CAD. In this research, a function is defined inclusively as the combination of the following aspects: (1) design intent, which is an abstraction of the purpose of design elements; (2) the flow of energy, material, and/or information transformation; (3) the behavior of a product to do something. It is true that functions could be defined flexibly and subjectively according to the application and need. It is not the focus of this research to standardize the "contents" of functions encountered in engineering design as they are mostly context-dependent. Rather, this research treats function as a knowledge unit and is aiming at introducing functional features that collectively represent such functional knowledge of product design with associative abstract geometry features to capture the principle of the design knowledge consistently, including the underlying physics.

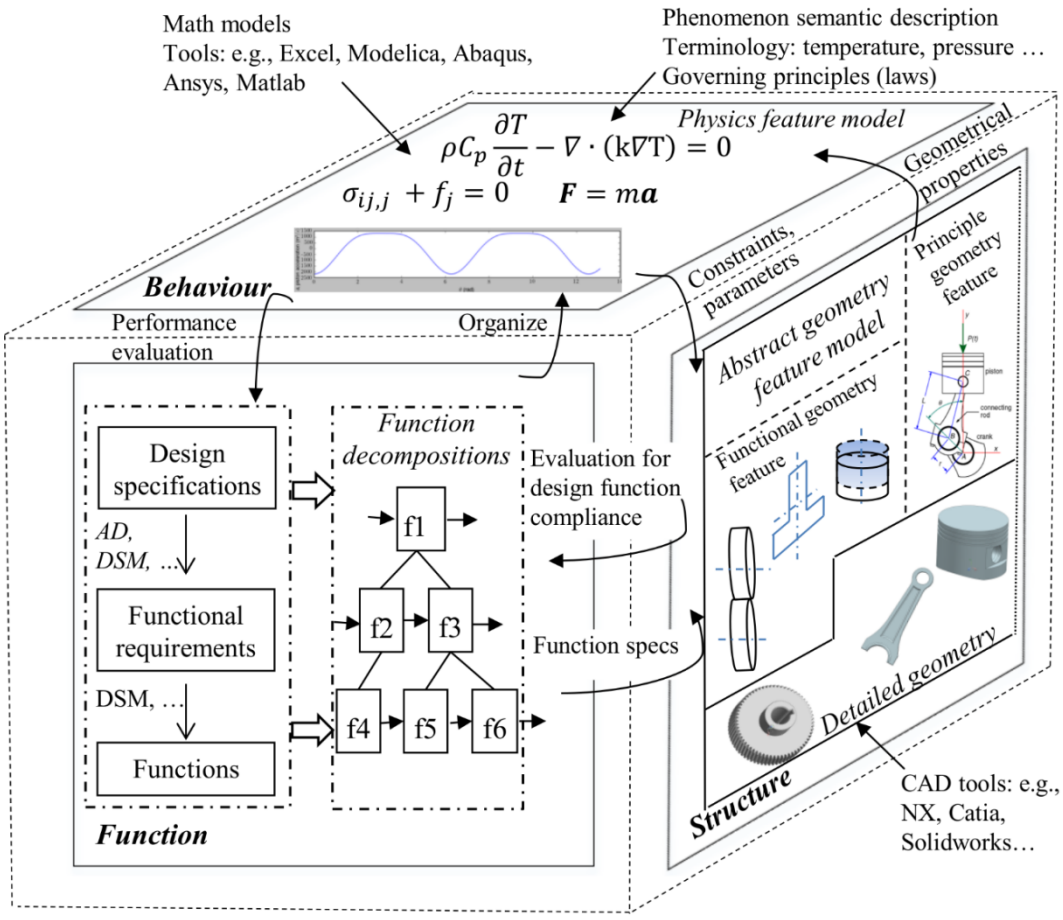


Figure 27 The functional feature modeling cube

Functional feature is a feature type that generically integrates the functional design intent, engineering physics, and product geometric model in a consistent functional-physical modeling approach. Figure 27 gives an overview of the functional feature modeling framework, i.e., to connect functional design, behavior modeling, and structure design together with functional features. Functional feature is not a concrete but abstract semantic definition that allows a way of design thinking to represent the associations among the design forms, functions and behaviors in the CAD systems. It acknowledges the role of physics in determining the product behaviors, the evolvement of design geometries from abstract to detailed form, as well as their connections with functions.

In terms of the functional design, functions are derived from functional requirements that are deduced from designs specifications. Those functions might not be primitive enough and function decompositions are required to further break down some of the functions into smaller granularities. The resulting functions could be used to organize the mathematically representable and clustered constraints used in the behavior modeling of the design artifacts. There are various sources of constraints, e.g., from the physics of the design that are depicted by the physics feature. Mathematical modeling is required to evaluate the performance result with the clustered sets of constraints, which can be done through tools like Excel, Matlab, Ansys, etc., depending on the nature of the problem. The results provide performance evaluation for the functions, creating a feedback loop to carry out the process iteratively (Figure 28).

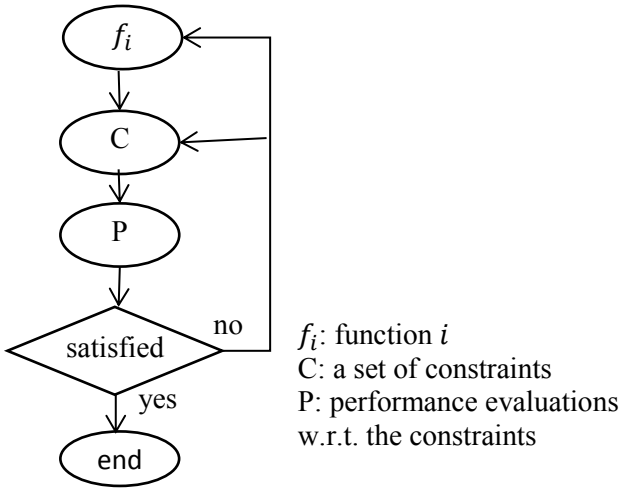


Figure 28 From function to behavior in the constraints' perspective

By the same token, function specifications are passed to the structure design, i.e., the designed structures need to conform to the functional specifications. In the structure regime, abstract geometry feature is proposed to capture the underlying functional-physical part of the design geometry (details follow in later sections). The structure provides the geometric data to the behavior modeling, for example, the domain or boundary of the design input to Computer-

aided Engineering (CAE) analysis. In return, the behavior modeling provides some constraints and parameters for the modifications and improvements of the design structure.

5.3 Semantic definition

The class diagram of functional feature is shown in Figure 29. Parameters in functional feature have different types, i.e., function parameters, performance parameters, physics parameters and geometry parameters. Detailed definitions of the above-mentioned parameter types could be found later. Physics feature is used to model the behavior of the product that is dictated by the underlying engineering physics. Constraints are embedded in the functional feature in the forms of relations among parameters as well as geometric elements, i.e., geometric constraints and dimensional constraints. Note that some of the dimensional constraints might be carried out in the form of relations in the geometry related parameters.

Abstract geometry feature is proposed to manifest the engineering/physics principles and functional considerations underlying the design. Abstract geometry features could be used in the conceptual design to represent the skeletons of the design. The semantics of functional feature provides an abstract interface from which different application level functional features could be defined. In the sense of Object-Oriented Approach, functional feature serves as an abstract parent class from which more detailed children classes could be derived and reuse the predefined defined interfaces for operating object attributes and inter-object communication.

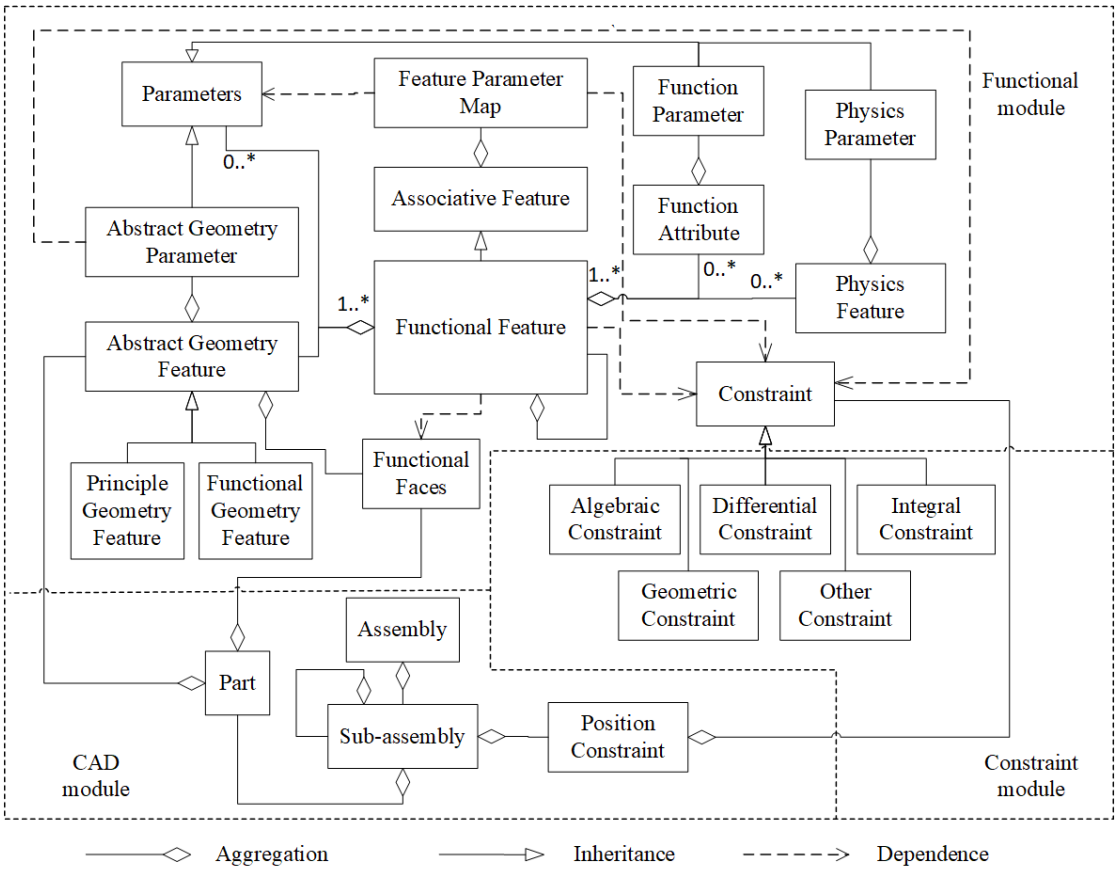


Figure 29 The UML diagram representing functional feature

5.4 Behavior modeling with physics feature

Behavior is defined by "sequential changes of states of a physical structure over time" in Habib and Komoto (2014). The behaviors are governed by certain physical laws, which are modeled by some mathematical models. Just to name a few, Newton's second law of motion, Fourier's law of heat conduction, Conservation of energy, etc. In turns, such mathematical models could be used to model the behaviors of the system under different conditions. Different problems are described by different states. For example, in linear elasticity problems, the state could be the displacement u of the physical domain; in heat conduction, the state could be the temperature T; in fluid dynamics problems, the states are usually chosen to be the velocity field v and the

pressure field p. Other quantities could be deduced from certain states, for example, in linear elasticity problems, strain tensor could be calculated based on the displacement field u, which could further be used to get the stress tensor. The states of the physical structures are subject to the behavior of the system with different input, e.g., initial and boundary conditions which is further governed by the underlining physics modeled with mathematical equations.

Physics feature, in the form of named variables and a set of mathematical equations describing the physics phenomena, is proposed to model the behavior of the design artifact. It contains information related to the physics/phenomena context involved in the design, for example, a mathematical model that describes a physical phenomenon, engineering properties that affect the design choice, etc. Note that one mathematical model could be applied to model different physical phenomena. It is not enough just to represent the physics related information to the design, there are engineering tools available for such behavior modeling, the result of which will be transferred to physics feature and further utilized by downstream design activities. For example, Modelica could model the dynamic behavior of the technical systems consisting of components like mechanical, electrical, fluid, thermal, hydraulic, control, etc., the systems of which that are described by differential, algebraic, and discrete equations. Some behaviors are described by partial differential equations, which required more advanced methods, for example, CAE, mostly solved with finite element method (FEM) and Computational Fluid dynamics (CFD), mostly solved with Finite Volume Method (FVM). Engineering tools are also available for modeling such behaviors, for example, Ansys, Abaqus, Fluent, OpenFAOM, dealii, etc.

It is not the goal of the current chapter to research tools for behavior modeling but to point out that with the understanding of the physics involved in the design process, such engineering knowledge should be readily available in CAD such that the management of the interaction

between the product structure (in the form of CAD model) and design considerations related to the engineering physics could be achieved.

5.5 Abstract geometry feature

5.5.1 Overview

Abstract geometry feature is a new concept of geometrical entity proposed to support functional feature modeling. Different from the physical geometry that represents the product final shape, which is solid, abstract geometry feature is a form of geometry that manifests engineering relational dependencies, performance characteristics, engineering principles, and functional behaviors. The key difference between the abstract geometry feature and the geometry nowadays constructed in CAD system is that abstract geometry feature is not restricted to the geometrical topology, i.e., the geometry constructed in the feature-based CAD system need to be manifold, whereas abstract geometry feature could be non-manifold.

The semantic definition of the abstract geometry feature is shown in Figure 30. Abstract geometry feature contains information like references, geometric entities like volumes, faces, and skeleton geometries, parameters, and constraints/relations.

In the current research, abstract geometry feature is further categorized into principle geometry feature and functional geometry feature, depending on the layer of abstraction. It could be either more functional oriented or physics principle oriented. Principle geometry feature is the geometry that manifests the engineering/physics principles underlying the design, for example, crank-slider mechanism. Functional geometry feature, although still abstract, is a form of geometry that manifests the functional shape of the design.

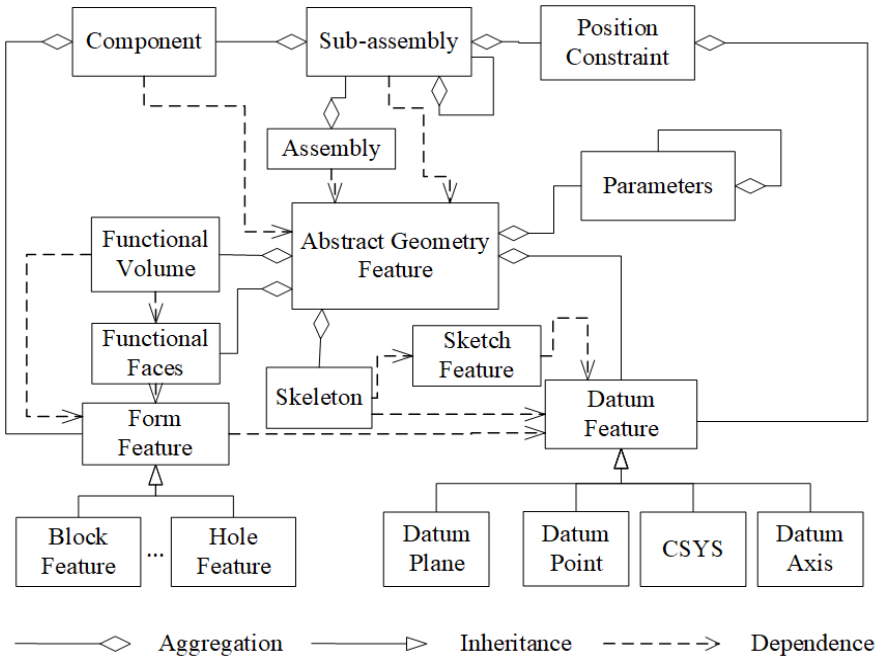


Figure 30 The semantic definition for a generic abstract geometry feature

Abstract geometry feature is associated with the behavior of the design and implies a certain generic shape and could be used to support reasoning with the attached attributes that enrich its semantics. The effectiveness of the abstract geometry is not restricted to a component. It can be used in an assembly structure and at the product level. For example, the skeleton structure of an abstract geometry feature could be used as the top-level guidance to position different parts in an assembly structure.

5.5.2 The need of abstract geometry feature

Due to the fact that engineering information regarding the design is usually incomplete during the early design stages, the design artifacts normally evolve progressively in the process of engineering design with information enrichment. However, the limited numbers of constraints enforced in the early design stages continue to be influential throughout the whole design process. Thus it is crucial to associate geometries of different design stages to provide a holistic

view of the design evolvement. For example, in the design of spur gear, with the given power and space constraints, the principle geometry of spur gear would be a circle that denotes the pitch circle. Note that pitch circle is not a real but imaginary circle on which most engineering calculations in gear design are based. One of the functional geometries is the extension of the principle geometry with gear tooth in 2D that contains more information like the number of teeth, tooth thickness, addendum, dedendum, etc., which captures the motion of the gear in a gear pair. The further evolved detailed product geometry would also contain information like face width with 3D solid representation. The evolvement of the geometry enrichment represents the progress of design with adding design details.

Abstract geometry feature supports design thinking and could be extracted or abstracted from parts. The representation of abstract geometry feature ranges from 0D (e.g., a mass point) to 3D. The "functional faces" of Schulte et al. (1993) could be included in the definition of abstract geometry feature. Abstract geometry feature, as the functional concepts carrier, is flexible. The flexibility ensures that it cannot only be used in a single part design but also in the product-level design.

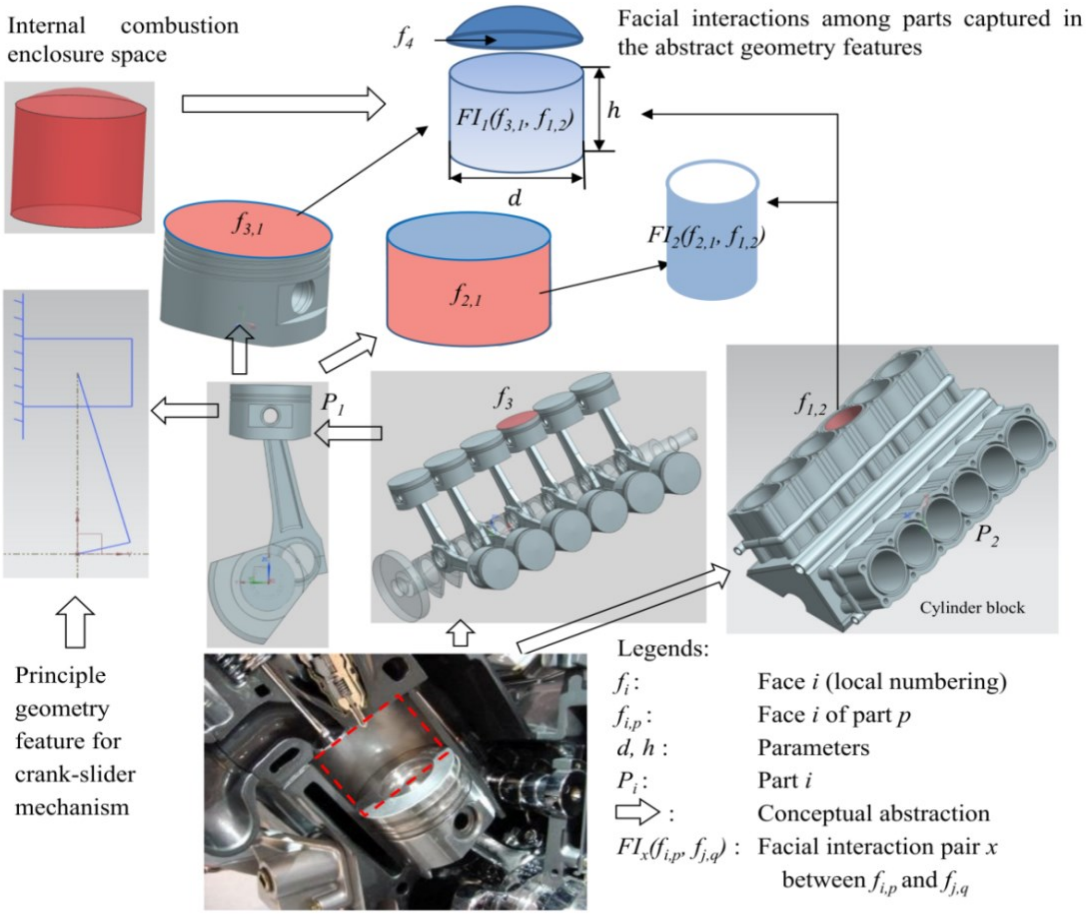


Figure 31 An inner combustion abstract geometry feature with facial interactions

An example of inner combustion abstract geometry feature is given in Figure 31. The bounded volume is the inner combustion principle geometry feature, in which complex combusting phenomena could happen. The inner combustion principle geometry feature is associated with inner combustion functional feature and its bounding faces, i.e., functional face $f_{3,1}, f_{1,2}$, and f_4, on the one hand, enclose facial interaction to form the volumetric principle geometry feature, and on the other hand, are linked to other abstract geometry features with geometric parameters, e.g., h and d, and non-geometric parameters, e.g., compression ratio, temperature and pressure within the chamber. Facial interaction F_{I2} $(f_{2,1}, f_{1,2})$, describing the

sliding of a piston within a cylinder block, could be further associated with the assembly features that determine the assembly requirements of the cylinder block and piston.

Moreover, Figure 31 shows the abstraction and extraction of functional faces, real (e.g., $f_{3,1}$) or imaginary (e.g., f_4) from parts to form facial interactions. It is natural that the process could be reversed, i.e., the materialization of those functional faces into real parts. That is to say, abstract geometry features with functional considerations will be eventually materialized in the detailed design stage where the functional faces evolve to different parts. For example, the cylindrical functional face $f_{2,1}$ evolves to the inner surface of a cylinder block, and the functional face $f_{3,1}$ evolves into a piston, of course with the convergence of some other functionalities into the piston part, e.g., from the principle geometry feature for a crank-slider mechanism.

Design process progresses through different stages. With the understanding that abstract geometry features capture the generic shape of the design artefact in its early stages and affect the design choices of the later stages, it is natural to add associativity to this process, i.e., associate the geometry used in the different stages of the design, including abstract geometries and physical product geometries, by means of associative feature modeling (Ma and Tong 2003; Ma et al. 2007; Ma 2013) with consistent parameters and constraints association management.

5.6 Constraints and parameterization of functional feature

5.6.1 Constraints and parameterization in engineering design and CAD

Constraints in the engineering design state the conditions that need be satisfied for the design to be viable and help to reduce the feasible solution space. Constraints are imposed on the design from different sources. For example, for the consideration of structural integrity under some

certain working condition, design artifact should not have the engineering stress value that is beyond the yield stress of the material. For another example, due to the restrictions of the manufacturing capability, a design artifact, e.g., slotted liner, might be required to have certain slot patterns instead of random slot patterns even when the slot patterns are, for example, "optimized" from topology optimization. Constraints in the design process, in general, manifest themselves in the structure of design artifacts and naturally in their CAD models, which, during the CAD modeling procedure, need to be determined by engineers, the constraints mapping of which usually lack a systematic approach.

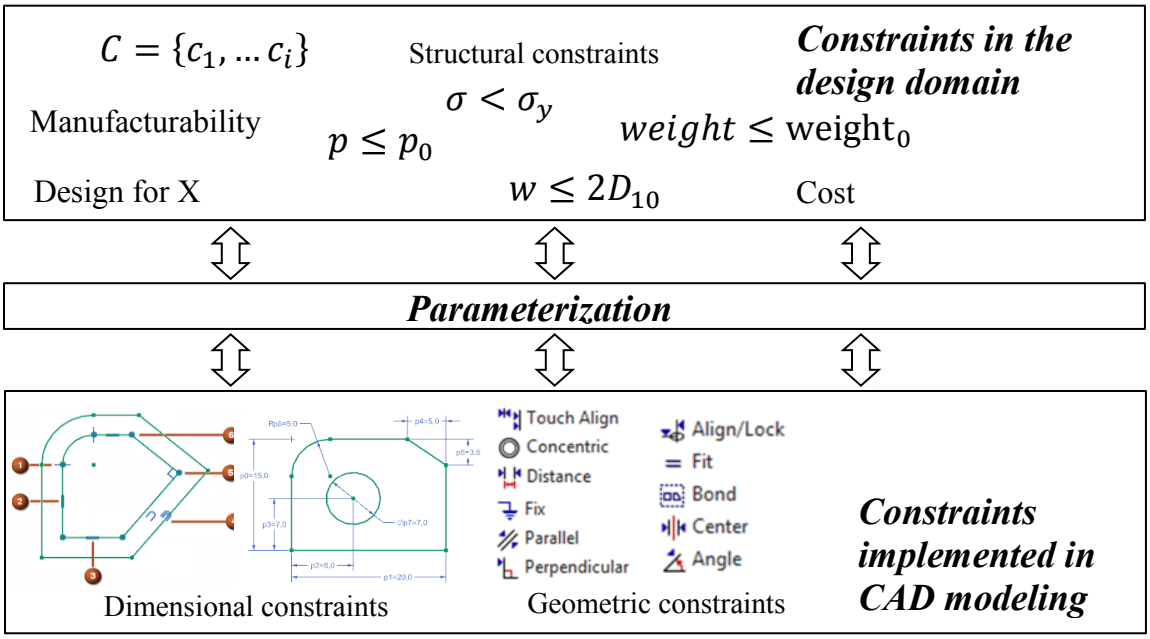


Figure 32 Parameterization to link constraints in the design and CAD

Constraints in CAD are primarily spatial constraints with different types, i.e., geometric constraints, dimensional constraints, and assembly constraints. Geometric constraints, for example, vertical, horizontal, and parallel, restrict the relative position of sketch objects with respect to a reference, which in itself could also be a sketch object. Dimensional constraints determine the size of a sketch object or form feature, for example, specifying the width and

length of a rectangular. Parameterization is what links constraints in the design and CAD modeling domains, see Figure 32.

Table 4 Examples of assembly constraints (Adjusted from Siemens NX documentation).

Constraint	**Description**
Align	Align two axes
Angle	Specifies an angle between two objects
Bond	Constraints objects together such that they move as a rigid body (without deformation)
Center	Center one or two objects between a pair of objects, or centers a pair of objects along another object
Concentric	Constraints two circular or elliptical edges so that the centers are coincident and the planes of the edges are coplanar
Distance	Specifies the 3D distance between two objects
Fit	Constraints two objects with equal radii
Fix	Fixes a part at its current position
Parallel	Defines the direction vectors of two objects as parallel to each other
Perpendicular	Defines the direction vectors of two objects as perpendicular to each other
Touch align	Constraints two components so they touch or align with each other

As mentioned above, the goal of assembly design in the current CAD system is to put components in their rightful position and orientation. When assembly constraint is applied to a component, its degree of freedom (DOF) decrease so that the component will get to where it is supposed to be. An object's location and orientation are completely determined when three translational and three rotational DOFs are fixed. Table 4 shows assembly constraints available in *Siemens NX*. Under the hood, it is achieved by multiplying model coordinate of the component by a series of transformation matrices in terms of the homogeneous coordinate (4×4 matrix), which is the product of rotation and translation matrix. Each component and subassembly has its own model coordinate. For example, if one wants to make two planar faces belonging to two different parts align with each other, it requires their normal vectors to be aligned (or reversed), and the inner product of the normal with a vector formed by two points in these two planar faces to vanish. To achieve this effect, transformations need to be applied to the mating part. With the assembly constraints provided in CAD system, engineers don't have to calculate the transformation matrices manually. This approach is more intuitive and user-friendly.

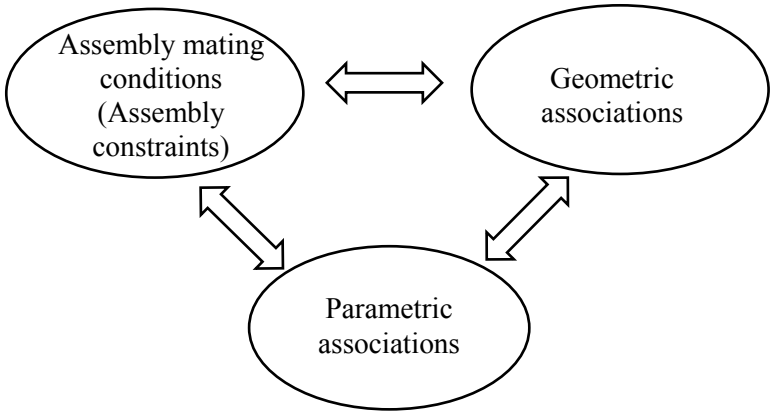


Figure 33 Associations between parts

Mating conditions, or assembly constraints, impose associations between different components, see Figure 33. For example, with a hole and a shaft one can require their center

lines to be aligned. Implicitly it requires the diameter of them to be within certain tolerances and fits. Here it could be observed that there is a geometric association between the inner circular surface of the hole with the outer circular surface of the shaft. This geometric association also incurs a parametric association, i.e. their diameters. The geometric and parametric association, however, are between parts, not within a single part. Sometimes it is hard to say which comes first, the associations or the mating conditions. Nevertheless, different knowledge entities could be formed as the controlling elements.

Parameters are not standalone. They should be organized in a meaningful way such that engineering semantics could be associated with a parametric control both geometrically and non-geometrically. For example, in the design of a thick-walled cylindrical pressure vessel (Figure 34), the geometrical design parameters, e.g., thickness t and radius of the pressure vessel r, are directly related to the non-geometrical parameters, e.g., stress σ under given pressure p. The stress is a physics quantity that, although could not be "drawn" in the product geometry, is associated with product geometry in a functional manner, i.e., to maintain the structural integrity of the pressure vessel under the working conditions.

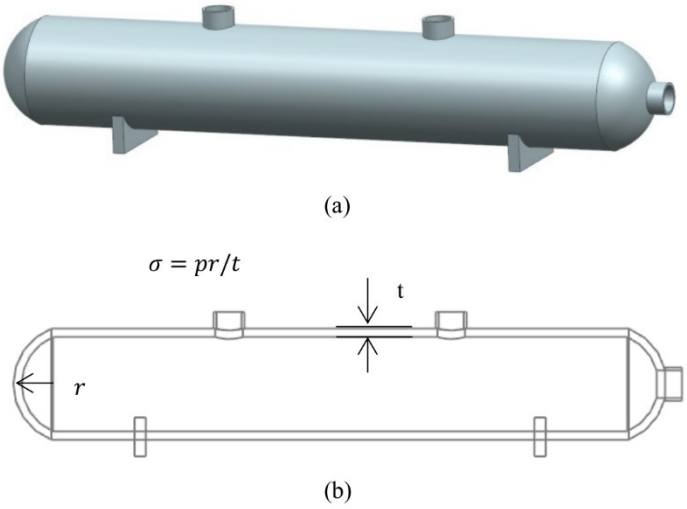


Figure 34 Geometric and non-geometric parameters in the conceptual design of a pressure vessel

5.6.2 Constraints and parameterization in functional feature

Constraints in different forms, i.e., continuous, discrete, and mixed constraints (Qureshi et al. 2010), are embedded in the functional feature. On the one hand, in the geometric representations of abstract geometry features certain kinds of geometric or dimensional constraints are applied, e.g., the constraints applied in the working faces or design skeleton. On the other hand, engineering design considerations impose some forms of constraints on the parameters of a functional feature from, for example, physics feature modeling and conceptual mapping from a high-level functional model. i.e., constraints from different aspects are handled separately but integrated into functional feature modeling.

With the understanding that a lot of parameters or variables are involved in the process of engineering design, it is crucial to have a systematic approach to manage those parameters. Functional feature modeling provides parametric management that is different from the traditional flat manner. Parameters in a functional feature, in general, could be geometric and non-geometric. To further categorize parameters of the functional feature, it could be seen that

some of them are used in describing the functional requirement of the design, which are referred to as *function parameters* (Bluntzer, Gomes, and Sagot 2008), or measuring the performance of certain functional features, i.e., *performance parameters*, or defining the physical phenomena, i.e., *physics parameters*, as well as describing the geometric entities, i.e., *geometry parameters*.

The roles of parameters might be overlapped. For example, a function parameter is possible to serve as a geometry parameter if it controls certain geometric shape directly. Such overlap is easy to handle as users could define one parameter based on the other with equality constraint, or use the existing parameter directly to, e.g., construct the geometry.

The enriched engineering semantics of parameters in the design can be mapped to the parametric change consistently with appropriate constraints management and feature definitions. With these, it could be seen that functional feature parameters could interact with parameters in the different level of the design. Figure 35 provides a holistic view of the relations among different parameters.

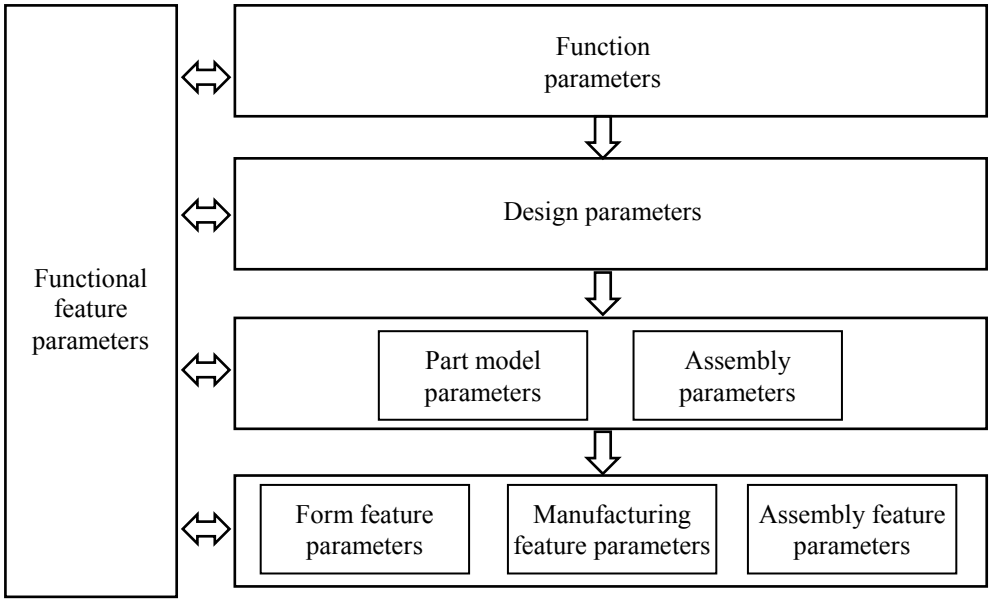


Figure 35 A holistic view of the relations among different parameters

5.7 Design with functional feature

Functional feature modeling serves as an intermediate layer between engineering functional design, which is mostly conceptual and declarative, and current CAD modeling, which is procedural, as is shown in Figure 36. In conceptual functional design, engineers construct functional models comprising multiple building blocks that describe what the product does with functional decomposition and morphological composition (Pahl et al. 2007; Frillici, Fiorineschi, and Cascini 2015). Functions are broken down, behavioral principles examined, and structure concepts proposed.

The interrelations among function aspect, structure aspect, and behavior principle aspect could be modeled with approaches like QFD, DSM, AD (Suh 2001; Torres et al. 2010; Eppinger and Browning 2012), the information of which can be embedded as attributes and passed down to the functional features. The decomposed functions make up the function attributes in the functional feature. The structure concept, which might be abstract, partial, or premature at this stage, could be represented with abstract geometry feature. Behavior principle is captured by the physics feature. Functional specifications, principle parametric map, and constraints compliance are passing down into the functional feature modeling layer, handled separately with different components in the functional feature.

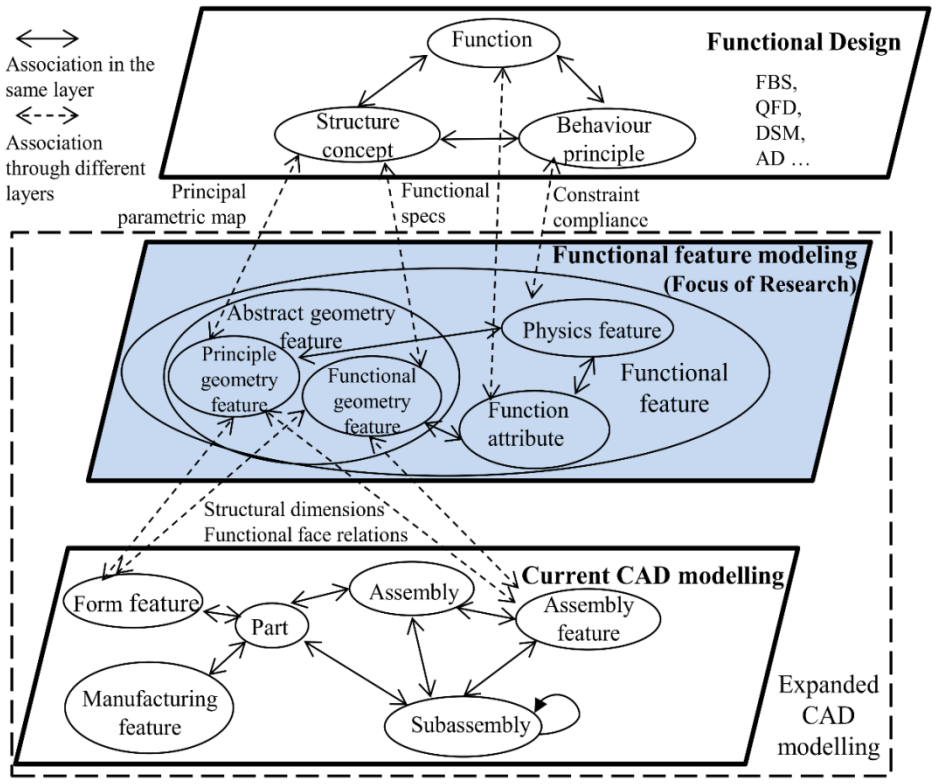


Figure 36 Functional feature layer supporting functional design with CAD

In functional modeling, a set of functional features could be built. Since not all information is available at this stage, a detailed geometric description is unavailable. However, an "abstract" form of geometric description is obtainable, which on the one hand guides the development of the detailed geometry constructions, and on the other hand, restricts the solution space of the possible structural description of the design. The physics considerations (captured by physics features) are associated with the abstract geometry feature, providing constraints. In short, the information passing down to the detailed CAD modeling is the structural dimensions, volumetric, and/or functional face relations, and/or skeleton and references geometries from abstract geometry feature, as well as the constraints dictated by the physics considerations.

With the establishments of abstract geometry features, detailed CAD modeling can be carried out, the modeling procedures and parameterizations with which should be carefully

planned out (see the previous chapter). By acknowledging the interdependencies among feature operations, users need to plan out the CAD modeling procedure to fit for functional feature modeling. Once the functional feature support is embedded in the CAD modeling, functional interfaces with dedicated Graphical User Interfaces (GUI) can be created to offer end users a cleaner UI to manipulate the design. Such user interfaces allow users to change the necessary functional parameters of the product without considering the detailed CAD modeling procedure.

5.8 Chapter summary

This chapter gives an overview of functional feature framework. Different building blocks of functional feature concept have been introduced in the current chapter, including physics feature, abstract geometry feature, and constraints and parameterization. This chapter is an attempt to instill functional considerations into CAD-related design activities. This chapter lays the foundations from which future chapters will continue to further enrich the concepts and demonstrate the use cases.

6 CAD Modeling with Abstract Geometry Feature

6.1 Chapter introduction

With the theoretical foundation laid down in the previous chapter, this chapter gets into a more in-depth discussion of applying abstract geometry feature in CAD modeling procedure. Abstract geometry feature is a key concept in functional feature modeling and it is worthwhile to further develop the concept. The interest of this chapter lies in the details of how to construct CAD models that are robust enough to capture functional design knowledge. The results of our approach could be integrated into model-based system engineering process to construct the valid and robust CAD models. The proposed approach in this chapter can be seen as a top-down modeling method. The "top" in here means the modeling of function-driven abstract concept carriers, i.e., abstract geometry features; and the "down" is the detailed CAD modeling, including part modeling and assembly modeling.

The resulting CAD models built with the method to be proposed in this chapter are functionally robust because functional considerations of design, manifested by function concepts carrier - abstract geometry features - are taken as modeling guidance with geometry associations, proper parameterizations, and constraints management. The modeling of the detailed CAD geometry is based on the synthesis of abstract geometry features, which in turn reflects design functionalities. Functional changes could be traced to abstract geometry features, or the relations among them, and then to the detailed CAD models. The traceability of functional changes into detailed CAD model makes it easier to carry them out.

The current chapter is organized as follows. After a brief introduction in Section 6.1, Section 6.2 discusses that abstract geometry features can be used as functional concept carriers. Following that section 6.3 gives a schematic modeling procedure with abstract geometry features. Section 6.4 presents an example to demonstrate in detail how to model with abstract geometry features. Section 6.5 provides a discussion of the proposed approach and compares it with an existing method. The last section summarizes this chapter.

6.2 Abstract geometry features as functional concepts carriers

According to Roy and Bharadwaj (2002) Part Function Model (PFM), faces of the part could be connected to part function relations. In the current research, abstract geometry feature is proposed to handle the representation of geometric elements of functions in CAD. Abstract geometry feature is a functional design concept carrier with an abstraction of the key characteristic shapes of the design artifacts. It provides, on the one hand, a suitable form of geometry for conceptual design and, on the other hand, guidance for modeling of the detailed design geometry. Recall that abstract geometry feature includes principle geometry feature and functional geometry feature, depending on the functionality it serves. It could be represented with both manifold and non-manifold geometries, including volumetric, functional faces, skeleton geometries, and references. Abstract geometry features share similarities with conventional form features, i.e., they both need to have references, relevant geometric entities, parameters, constraints, and feature ownerships. It could be made into a template where other knowledge elements could be attached as attributes.

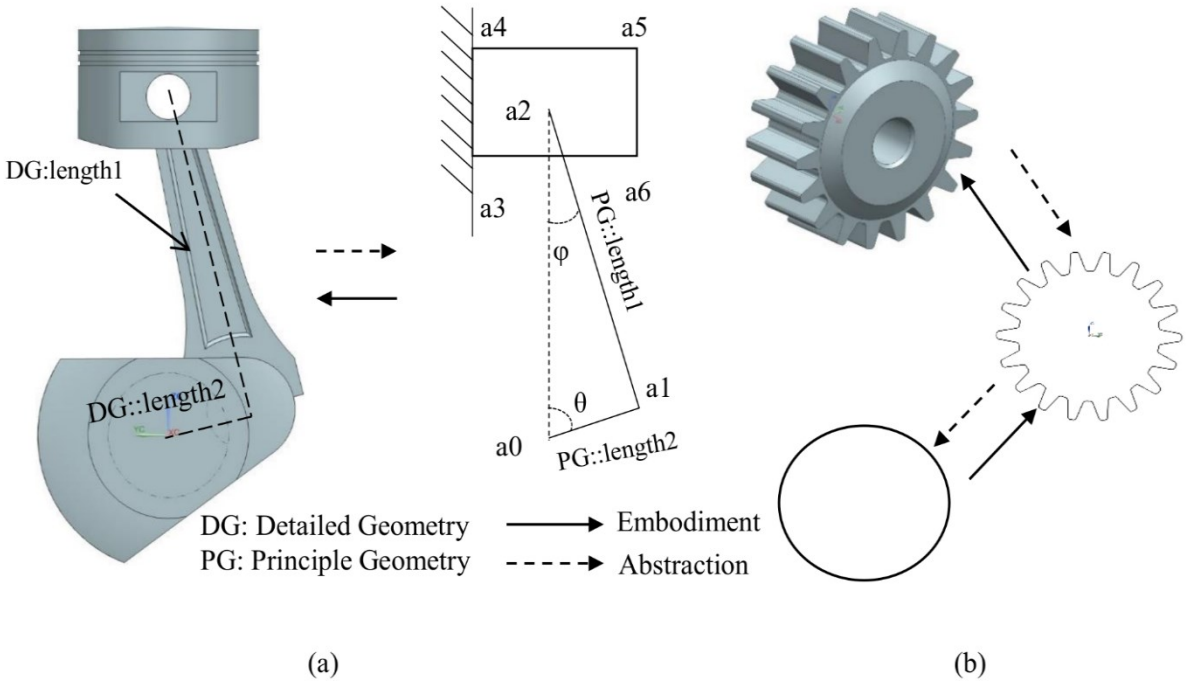


Figure 37 The abstraction and embodiment of geometries of different LODs in the design process

An abstraction process needs to be carried out to reach abstract geometry features. The abstraction rules are flexible and application dependent. Some general guidelines are: (1) abstract geometry feature captures the kinematic relations (physics) of the design (e.g., gear, slider and crank mechanism); (2) abstract geometry feature captures the general shape of the design (e.g., pressure vessel without thickness); (3) abstract geometry feature captures the functionally important shapes. With the characteristic geometry available, detailed and fully-fledged CAD geometry can be refined from the abstract geometry, the refinement could be done either manually or automatically with some programming. Figure 37 gives two examples of abstraction and embodiment of geometries of different levels of details in the design process. Attributes are attached to the abstract geometry features to enrich their semantics such that downstream activities could derive the necessary information from the object of abstract geometry features.

For example, the tube used in the slotted liner abstract geometry feature can be constructed without thickness; however, the thickness value could be attached to the abstract geometry feature such that in generating the detailed part geometry operation could be performed by reading the thickness value of the abstract geometry feature.

Constructed or extracted abstract geometry features could be stored in the feature library for future usages. Some abstract geometry features cannot be converted to detailed CAD product geometry straightforwardly. However, it does control some key dimensional constraints of the product geometry. Implementation wise, abstract geometry features are implemented as User Defined Features (UDFs) or User Defined Objects (UDOs). During the construction of the detailed part model, users can use them as regular features; the geometric and non-geometric elements can be referred to freely. Or use them as separate parts models from which other parts can refer to by using technology like, for example, *WAVE* from Siemens NX. *WAVE* can link, both associatively and non-associatively, geometry elements like bodies, curves, datum, faces, and points, between two parts and get information about the linked geometry and parts, including parameters and applied constraints. The functionalities of *WAVE* and expressions are adopted into the abstract geometry feature module.

6.3 A schematic modeling procedure with abstract geometry features

A brief schematic overview of the general modeling procedure is shown in Figure 38. It mainly consists of three major steps, i.e., functional analysis, abstract geometry features modeling, and CAD part geometry synthesis. The rest of this section will discuss the procedure briefly.

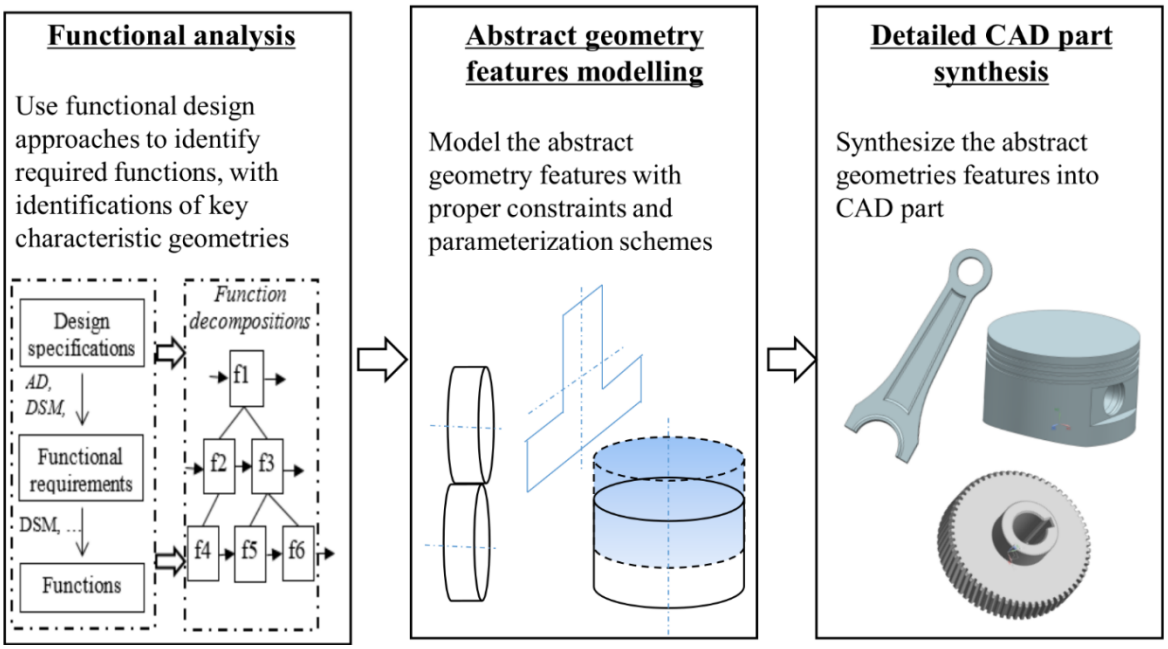


Figure 38 A schematic overview of the proposed general modeling procedure

6.3.1 Functional analysis of the design artifact

Approaches like QFD, DSM, and AD (Suh 2001; Torres et al. 2010; Eppinger and Browning 2012) are applied to identify the required functionalities for the design artifact. Domain knowledge could be elicited from experts or from existing documents/patents. Functional decompositions need to be carried out to decompose more general functions into smaller granularities. The resulting functions could be in the form of a tree-like structure. Identify the key parameters, both geometrically and non-geometrically. Build the relations among geometric and non-geometrically related parameters with feature parameter maps (Yin and Ma 2012). Identify the functional faces (Roy and Bharadwaj 2002) or other key characteristic geometries required to perform the functions.

6.3.2 Abstract geometry features modeling

Model the abstract geometry features that enable the design functionalities. References,

constraints, and parameterizations within the abstract geometry feature should be well organized accordingly. Then identify the relations among those abstract geometries spatially. For example, a few abstract geometry features might need to be formed within a single solid part. In this case, the spatial relations among the abstract geometry features need to be considered, again well constrained and parameterized, which is done, preferably, through their references.

In some scenarios, the spatial relations need to be determined with other design considerations. Parameters within abstract geometry features should be named meaningfully. The constructed abstract geometry features can be made into UDFs, or, if more flexibility is required, they can be programmed as UDOs, or saved as separated files. They could be placed into a feature library and are reusable for future design activities.

6.3.3 Detailed CAD part modeling

This is the stage that performs the modeling activities to construct the CAD model for detailed design. After identifying the abstract geometry features, it is often not easy to synthesize abstract geometry features into fully fledged CAD model with current CAD tools at hand. Ideally, the synthesis process is straightforward by simply combining the abstract geometry features together with proper positioning and additional feature operations. Rather, designers must make use of available modeling operations to construct the geometry where the abstract geometry features and their spatial relations are embedded, namely, one might not use the exact same modeling operations used in modeling the abstract geometry features, or make direct use of abstract geometry features, to model the corresponding geometry during the detailed CAD model construction. Nevertheless, the results of parametrizations and constraints identified above provide guidance for this stage.

6.4 An example of abstract geometry feature modeling

Design of a piston part is taken as an example in the current chapter. Note that the case study does not mean to be inclusive but to offer a demonstration of functional modeling of CAD with a few representative functional considerations. It can be observed from the case study that part faces are commonly used to reflect abstract geometry features. However, this does not mean it is restricted to faces only. As mentioned above, other geometric entities like solids, edges, vertices as well as constructive elements, such as sketch elements, datum planes, center point positions, feature dimensions, can be applied. Non-geometric entities like attributes, derived parameters and constraints can also be used.

6.4.1 Functional analysis of a piston

As discussed above, engineers should embed the functional considerations of the part being modeled in the CAD modeling process such that the resulting model is functionally robust. The main function of a piston used in an engine is *to transfer force* from expanding fuel in the cylinder to the crankshaft with the piston rod, the function of which could be further decomposed. It plays a central role in the overall function of an engine, i.e., to convert one form of energy, e.g., heat, to mechanical energy. It undergoes a linear motion incurred by the high temperature and high-pressure gasses burning in the combustion chamber. It, together with connection rod and crankshaft, helps to convert the linear motion to rotational motion. Moreover, to prevent the combustion gases from bypassing the piston, *sealing* need to be considered as well, which will be handled with the help of metal rings, or piston rings, around the piston. From the above discussion, it could be seen that in order to function well for the piston, it needs to have following functional faces

1. A functional face to interact with fuel, i.e., compressing and expanding. Henceforth denoted as f_1.
2. Grooves on which piston rings could be placed. These serve multiple functions. For example, those piston rings seal the combustion chamber to prevent gases from leaking to the crank and support heat transfer from the piston to the cylinder wall. Henceforth denoted as f_2.
3. A functional face to connect with the connecting rod, i.e., through piston pin. Henceforth denoted as f_3.

Note that the functional analysis examples listed above are not meant to be complete. In this scenario, a piston is not seen as a standalone object but is put into a context that it could interact with other parts of the whole system to perform certain functions. The key of the interactions lies in the geometry of the product, the functional faces, to be more specific in this case.

6.4.2 Abstract geometry features modeling

Based on the identified functions, abstract geometry features could be modeled, as is shown in Figure 39. For example, for f_1 the geometrical representation of the abstract geometry feature is a circular surface, and for f_3 a cylindrical surface, with corresponding references, parameters, and constraints. For example, it is clear that a circular surface could be parameterized by its diameter or radius and referenced by a coordinate system, and a cylindrical surface parameterized by its diameter or radius and its length and referenced by its own axial. There might be more than one ways to reference or parameterize abstract geometry features, depending on the requirements on the restriction of the corresponding degree of freedoms. Constraints could

be applied to build up the relations among the parameters. The fundamental is that each abstract geometry feature is self-contained in the sense it is properly parameterized and well constrained with appropriate references. Since parameters are named they are easy to be identified and changed if needed. The model should adhere to the functions such that once an upstream functional requirement is changed it should also be updated accordingly.

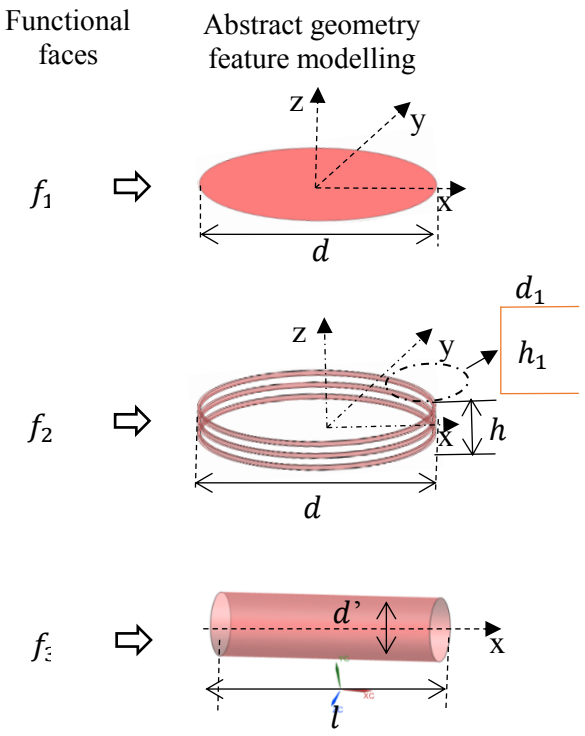


Figure 39 A schematic of from functional faces to abstract geometry feature modeling

The spatial relationships among abstract geometry features also need to be considered. For example, top land, the distance between the edge of the piston crown and the top side of the first piston ring groove, p_1, is a critical parameter in the design of the piston. The first piston ring is a compression ring. It requires a temperature range that needs to be compatible with its function. It is known that the value of p_1 is a compromise between different factors. For example, the piston is preferred to have low mass, which means to have a small p_1. However, p_1 also pertains to the function of the first piston ring, which is further related to the compression process, material, etc.

(GmbH 2012). For another example, the compression height, the distance between the center of the piston pin and the upper edge of the top land p_2 is also critical in the piston design. It needs to be as small as possible to have a low mass. However, if the value is too small, it will result in higher temperatures in the pin bore and high stress on the piston crown, which is likely to give rise to cracks in the pin bore or the piston crown (GmbH 2012). Such design considerations need to be transferred into the constraints among abstract geometry features and well parameterized, which in turn manifest into the detailed CAD model.

Since abstract geometry features are well constrained and parameterized, they are adaptable to new use cases. Different function attributes could be attached to abstract geometry features because the same abstract geometry feature could carry different functional concepts just like a mathematical model could be used to descript different physical phenomenon.

6.4.3 Detailed CAD part modeling

With the identified abstract geometry features, the next step is to synthesize them into the detailed model. When materializing abstract geometry features in the detailed model designers might not be using the same modeling operations used in constructing abstract geometry feature to model the corresponding parts with detailed geometry. Actually, there are also more than one modeling strategies to construct the detailed CAD geometry even with seemingly the same abstract geometry features. Certain entities of the abstract geometry features could be imported or linked into detailed CAD model to facilitate the model construction, e.g., geometry elements, parameters.

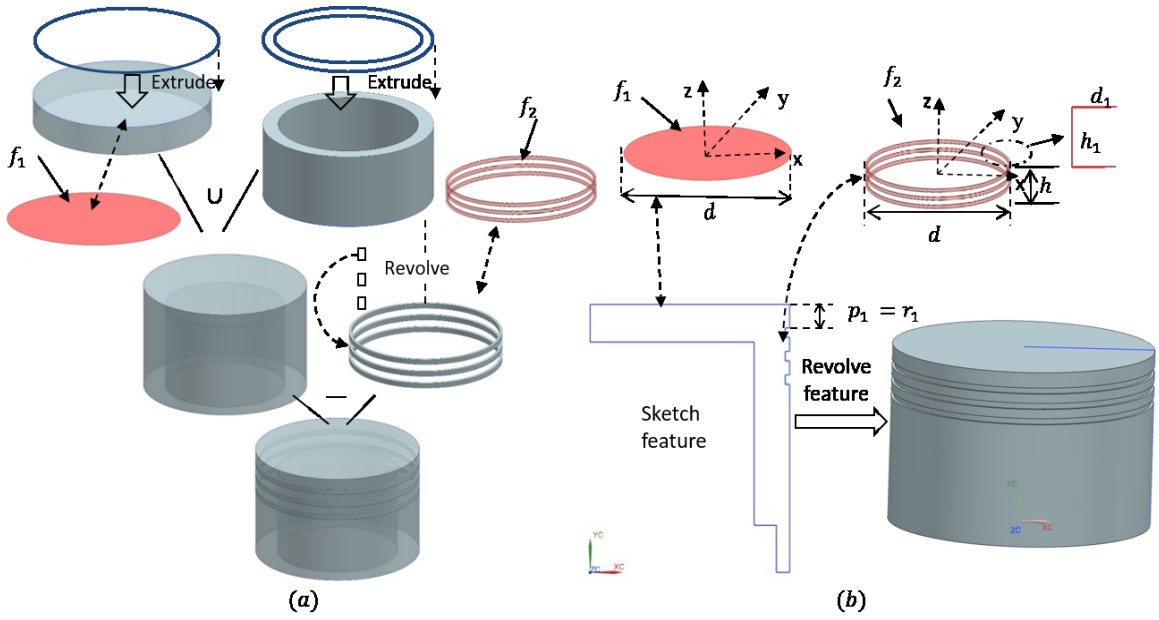


Figure 40 Two different approaches to synthesize abstract geometry features into detailed CAD modeling

For example, Figure 40 shows two examples of embedding two of the abstract geometry features into the detailed CAD model. In Figure 40 (a) it is done separately. The edge of face f_1 is used directly to extrude into a solid. f_2 is manifested by revolving three rectangular shapes. Proper positioning and *Boolean* operations are applied to combine the resulting solids together. Not all details of referencs are shown in the figure in order to save space. It is doable but not optimal in the sense that the process could be synthesized more organically. Figure 40 (b) gives a better example of synthesizing abstract geometry features into detailed CAD model. At least two different abstract geometry features are combined into one sketch such that a single revolve feature could build up the required intermediate model whereas the previous example demands many more feature operations. Moreover, the synthesis of abstract geometry feature into the detailed model might not seem to be straightforward. For example, functional face f_1 is not materialized by extrusion of a circular with a same diameter, as is shown in Figure 40 (a), but a revolution, as is shown in Figure 40 (b).

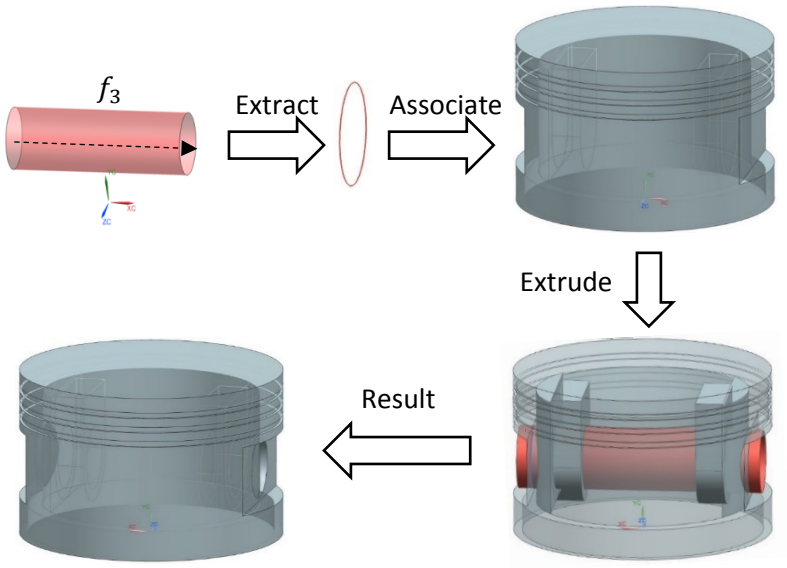


Figure 41 The manifestation of the face f_3 into the detailed CAD model

Parameterizations are applied properly such that the resulting faces have the same dimensions. The synthesis for f_3 could be carried out in a similar manner (see Figure 41). It is desirable to point out that other than synthesizing abstract geometry features in the part model, it could also be shown that abstract geometry features are often the key to associate different parts together. As is shown Figure 42, face f_3 serves as an interface among different parts and it indicates certain kinds of assembly constraints required to position the parts. The associativity requires that when f_3 is changed the relevant parts should also be updated accordingly.

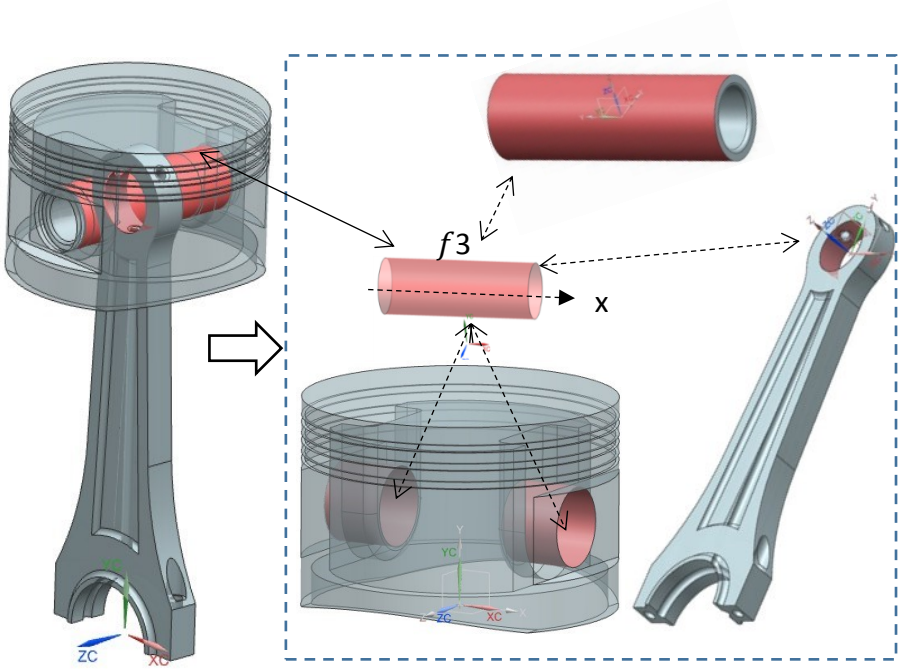


Figure 42 An abstract geometry feature used as the interface to associate different parts

As mentioned in the stage of abstract geometry feature modeling, the spatial relations among abstract geometry features need to be considered beforehand and they need to be manifested into the detailed CAD model. The manifestation is achieved through either geometry association or proper constraints and parameterizations, or the combination of both approaches. For example, the relationship predefined by p_1 is applied in Figure 43 to define the top land and p_2 to define the compression height. In sum, Figure 43 shows the schematic of the modeling process with abstract geometry features.

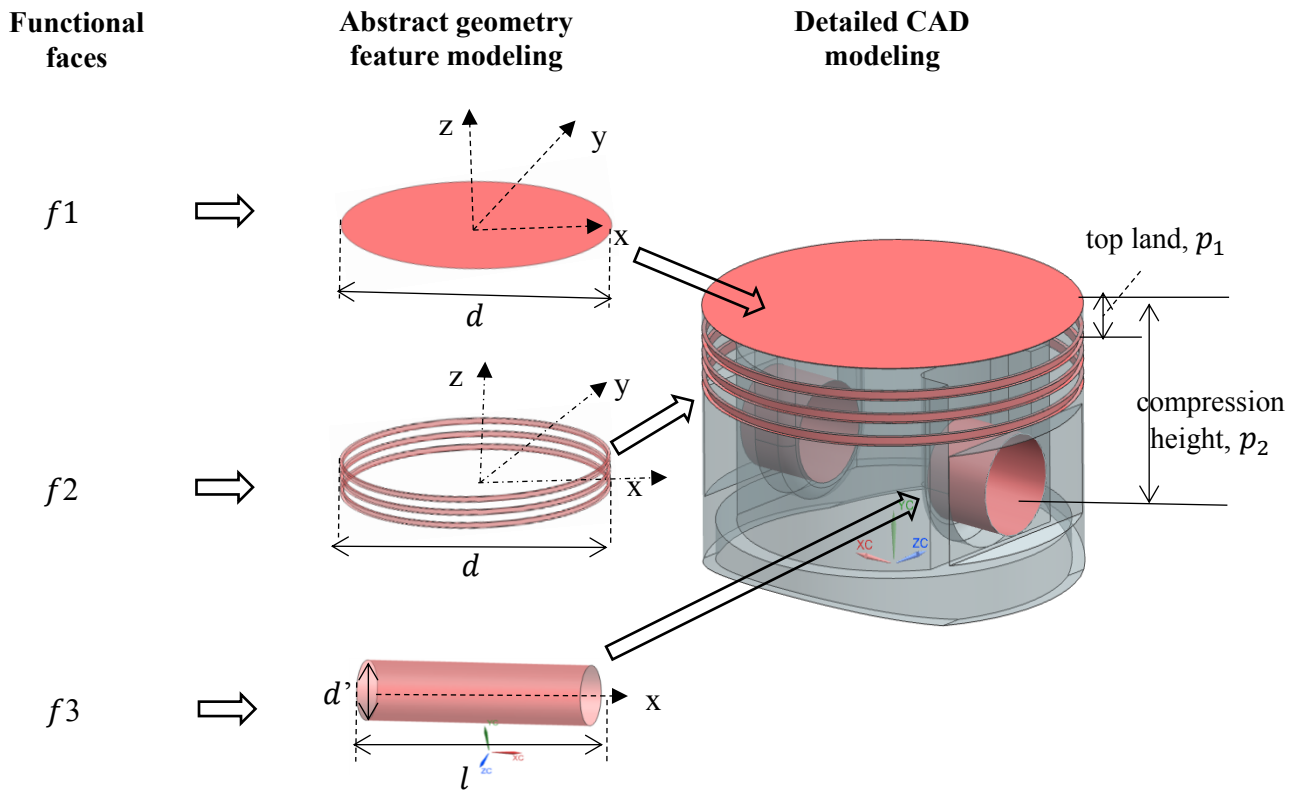


Figure 43 A schematic of the modeling process with abstract geometry feature

6.5 Discussion

Since both explicit reference modeling (Bodein et al. 2014) and our functional feature modeling approaches appreciate the importance of functions in the CAD modeling, detailed comparisons between those two will be given. Table 5 briefly organizes the key differences between explicit reference modeling and the proposed functional feature modeling with abstract geometry features.

First, both approaches emphasize on the uses of references. In explicit reference approach, the references are mainly referred to as those elementary parametric elements, which could be points, planes, surfaces, etc. On the other hand, the concept of references in functional feature modeling is broader in the sense that it includes datum coordinate systems, datum points, datum

axis, and datum planes. It seems that parameterization is not the focus in explicit reference modeling other than being embedded in the references. Parameterizations of the references, as well as other geometric entities, in abstract geometry features and functions are encouraged in the functional feature modeling, which are organized with feature parameter maps (Yin and Ma 2012).

In explicit reference modeling, constraints are categorized into mandatory and non-mandatory ones and they suggested using references instead of the current shapes to apply constraints in the non-mandatory cases whereas building features close to their primitives in the mandatory cases. Constraints in their approach mainly refer to geometric related constraints. Functional feature modeling includes both geometrically and non-geometrically related constraints; for example, geometric constraints, dimensional constraints, and assembly constraints are geometrically related constraints, whereas constraints applied to the functionally related parameters are non-geometrically related constraints. Constraints in functional feature modeling have richer engineering semantics.

In explicit reference modeling, solids for each function are constructed independently and then combined by using *Boolean* operations, which makes it unclear what they do when overlapping or disjoint functions exist in a given solid region. In functional feature modeling, since abstract geometry features, as abstract concept carriers, are not necessarily solid, *Boolean* operations alone are not enough. The current method is to choose the best approach to integrate or synthesize abstract geometry features into the detailed solid model construction with proper constraints and parameterization, instead of depending on *Boolean* operations alone. Thus, functions can not only be traced down to a solid region in CAD model but also in abstract geometry forms.

Table 5 Comparison of explicit reference and functional feature modeling approaches

	Explicit reference modeling approach (Bodein et al. 2014)	**Suggested approach**
References	• Parametric elements, e.g. points (instead of vertex), plane, or surface (instead of face) • Creating references for functional areas	• Datum coordinate systems, datum points, and datum planes • Referential parameters, as well as references, converted from sketch elements
Parameters	• Embedding in the parameterized references • Otherwise not clear	• Constraining reference elements • Representing dimensions of the abstract geometry features, functional parameters, and principle parameters, etc. • Feature parameter maps
Constraints	• Mandatory and non-mandatory constraints • Geometric related constraints • Creating features close to their primitive for mandatory constraints with implicit references	• Geometrically and non-geometrically related constraints • Containing constraints during design process as well as constraints involved in geometry creation in CAD • Parameterization
Geometries related to functions	• Functional areas, which are solids, resulting from functional analysis • Not clear what to do when multiple functions coexist in part of the same solid region	• Abstract geometry features • Both manifold and non-manifold geometry • Not necessarily solid, could be points, surfaces, volumes, etc.
Solid product geometry	• Applying Boolean operations on functional areas (solid only)	• Synthesizing abstract geometry features in the detailed modeling process

As indicated in (Bodein et al. 2014), it is difficult for a designer to apply a generic modeling concept during the design phase. Additional training is required before designers can apply their proposed CAD modeling methodology. It applies in the current case as well. Meanwhile, since simply building the shape of the product is good but not good enough, design thinking must also be instilled into the CAD practitioners and reflected in the model building process. CAD education in universities should not only focus on teaching the CAD software but also address the functional modeling methodology, which helps to build functionally robust CAD models.

The proposed method might sacrifice some easiness during the model creation but boosts functional knowledge capture and manifestation, and facilitates the design changes implementation. It might not be the easiest method to create the shape of the model but it strives to construct the models that are robust and ready for functional changes. On the other hand, CAD systems need to be enhanced to streamline the synthesis of abstract geometry features into fully fledged CAD model for detailed design based on the proposed modeling method.

6.6 Chapter summary

This chapter presents an in-depth and detailed description of a functional feature modeling method that entails how to construct robust CAD part model with abstract geometry features within functional feature modeling framework. A case study is presented to demonstrate the proposed method in an extensive manner. With multiple possible ways to construct the CAD model for a given product, a functional approach is believed to be effective to convey design intents. It could serve as a guideline for CAD practitioners to build functionally robust CAD model with smooth functional design change capabilities.

The main innovation of our approach lies in its function-oriented nature with a systematic modeling method. In particular, the proposed approach incorporates functional semantics into CAD models and narrows the gap between function-oriented design idealizations and procedurally-constructed CAD geometries. A systematic modeling procedure is presented with a detailed description of modeling with abstract geometry features. Ideally, designers could start from the functional consideration, trace to its abstract geometry feature representation, which is further linked to the materialization in the fully-fledged CAD model with associated geometries, parameterization, and constraints management.

7 CAD Modeling with Functional Features

7.1 Chapter introduction

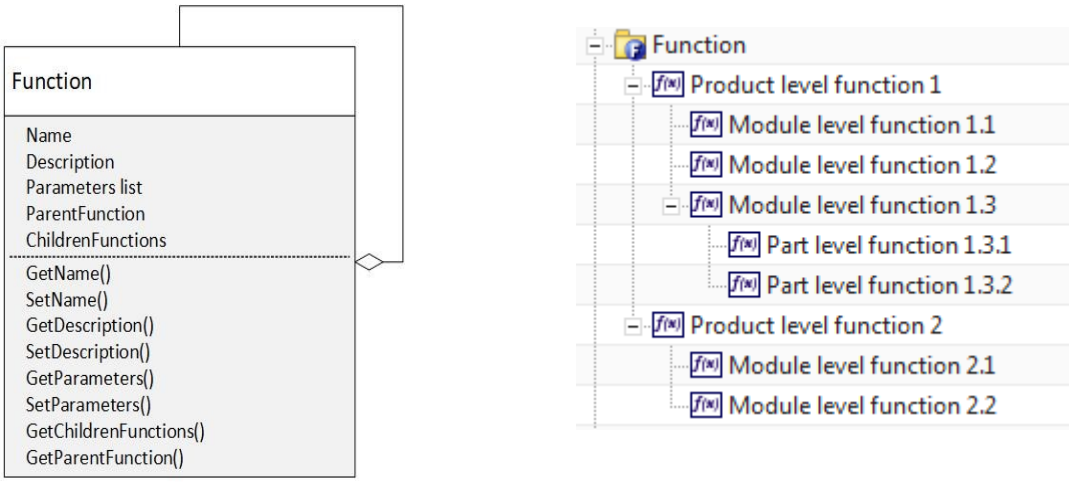
The previous chapter deals with the geometric aspects, abstract geometry feature modeling, of the functional feature modeling approach. It discusses how to construct the abstract geometry features and how to synthesize them into detailed CAD modeling. Thanks to its generic definition, the introduced functional feature modeling framework does not restrict the level of which it can be applied in the product design. This chapter would take a holistic view of the implementation of functional features in the different level of product design and development from functions' perspective. Functions can be roughly categorized into different levels, i.e., product level, module level, and the part level. All functions of smaller granularity come from product-level functions.

The rest of the chapter is organized as follows. Section 7.2 gives a general discussion on different levels of functions in product design. Section 7.3, 7.4, and 7.5 discussed the implementation of functional features at the product level, module level, and part level, respectively. This basically follows the logical order of functional decomposition. The last section summarizes the chapter.

7.2 Different levels of functions

From the literature review, it can be seen that functions in product design have hierarchical tree-like structures ((Pahl et al. 2007). A possible semantic definition and the resulting of functional decomposition of a product could be found in Figure 44. It could be clearly seen that the

attributes in the function include the name of the function, the descriptions, and references/pointers to the parent and children functions. It also provides methods to get and set those attributes. With this, hierarchical function structure for a product can be defined to include product-level, module-level, and part-level functions.



(a) Function class definition (b) tree-like function structure

Figure 44 Function definition and tree-like function structure

Produce level functions stem from customer requirements and other design specifications. Product-level functions are of high-level abstraction with a lot of unknowns. Functional decomposition is used to break down those high-level functions into smaller granularities, e.g., module-level and part-level functions. The categorization is mainly based on the functional consideration of the design. It is noted that this categorization is not clear-cut. For example, a product-level function might be fulfilled by a module, which indicates that the corresponding functional feature could also be categorized into a module-leve functional feature. For another example, there might be the case where a structure that must be assembled from multiple parts due to manufacturing constraints can be constructed into one part with the advances in

manufacturing technology. In this case, instead of a module-level functional feature, a part-level would be more appropriate.

7.3 Product-level functional feature implementation considerations

A product-level functional feature concerns with one or more product-level functions. A product-level function is a function defined, as its name suggested, at the product level. Product-level functions are the high-level functions that could be defined on a product. Correspondingly, the abstract geometry feature might be the simplest and most abstract, which might not be that intuitive. A product might contain many modules and parts. A product-level function, still, needs to be fulfilled by some structures within this product. The fulfilling structure might be one or more parts or modules.

Just as a single part might fulfill multiple functions, a single function might be mapped to multiple parts. In the case that one function is mapped to more than one part, there are one or more key geometries, like functional faces or volumes, for this function, which are the abstract geometry features that will be further mapped to multiple components. Multiple functional features could be defined at the product level (See Figure 45), corresponding to different functions defined at the product level. In addition, as has been discussed in Pahl et al. (2007), different working principle might exist for the same function structure. Note that the abstract geometry features of a functional feature for a product-level function might be mapped to one or more parts/modules. The defined constraints and parameters in this level are further transfer downward to module and part levels.

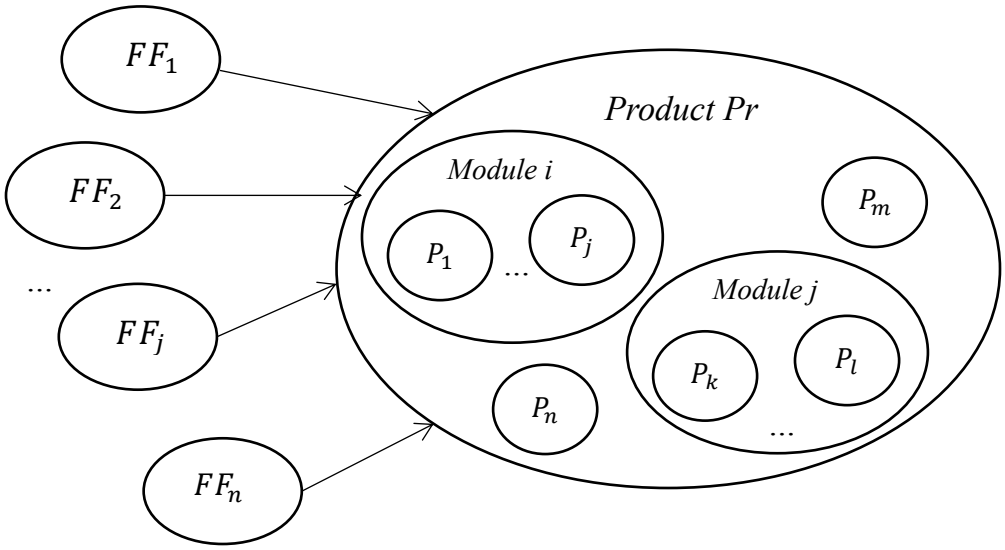


Figure 45 The part-level functional features schematic

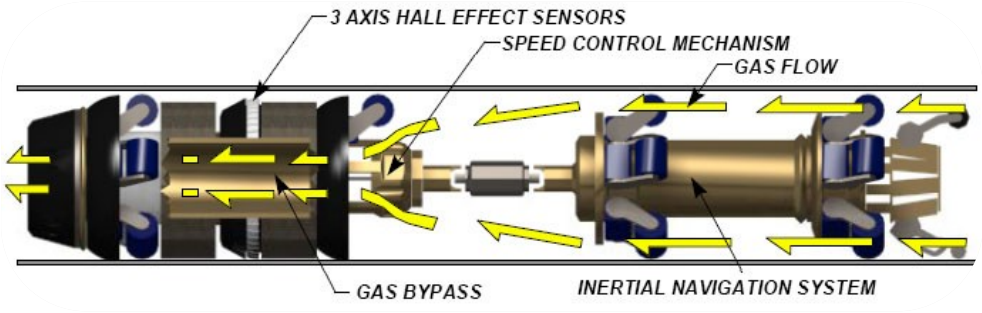


Figure 46 An example pipeline inspection gauge demonstration [5]

To make the concept of product-level function clearer, let's examine an example of a pipeline inspection gauge for natural gas pipelines from a student project[5]. A pipeline inspection gauge runs through the natural gas pipeline to inspect the integrity of the pipeline. The gas travels at a relatively high speed, around 10 m/s. However, due to the required resolution and the response of the sensor system, the maximum speed of the tool is specified at 3 m/s. A speed control system is needed to achieve this goal.

[5] Dayln Beazer, Michael Ross, Andrew Nielsen, Mark Staples, Pipeline Inspection Tool Speed Control, 2010, MEC E 460 Capstone Design Project

One of the product-level functions is, obviously, to control the speed of the pipeline inspection gauge such that it is no more than 3 m/s when it is traveling in a pipe. To achieve this function, one possible working principle is to change the flow area through the pipeline inspection gauge to change the thrust and thus the speed of the device. The bigger the open area, the less thrust force are applied to the pipeline inspection gauge. The logic train is: change the flow area => adjust the thrust => control speed.

There are many ways to change the flow area. The figure below listed a few possibilities. Conceptually speaking, all these alternatives provide the function to control the flow area. However, not all of them are viable when considering other design constraints. For example, space restriction, part counts, etc. The key here is that for the same function, control the speed, with about the same working principle, change flow area, there are different alternatives.

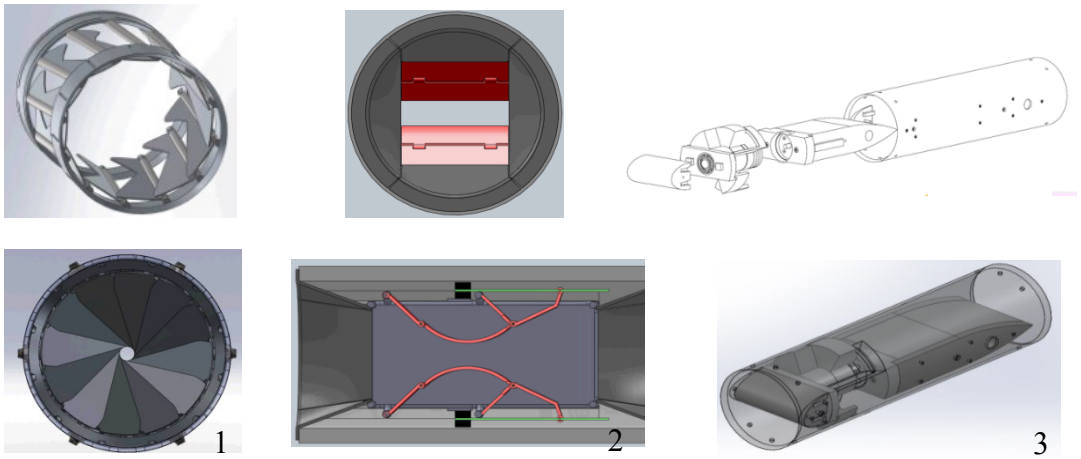


Figure 47 Different alternatives of design concepts to change the flow area[5]

The physics parameters involved include the pressure P, temperature T, speed of the fluid v within the pipe, and the friction. The pressure drop provides thrust. The pressure drop multiple by the blocked area and the drag are the thrust forces. The counter force primarily comes from friction between the pipeline inspection gauge and the pipe. The net force would be the

difference between the thrust and the drag, which provides the acceleration, positive or negative depending on the opening area. Of course, to control the speed a lot of more information is needed by either experiment or numerical simulation. For example, the friction is likely not constant for the different speed of the device. Neither is the thrust force of different opening area at different speed. The relation among the opening area, the speed of the device, the thrust, and the friction forces need to be determined before a controller can be made, which is out of the scope of this discussion.

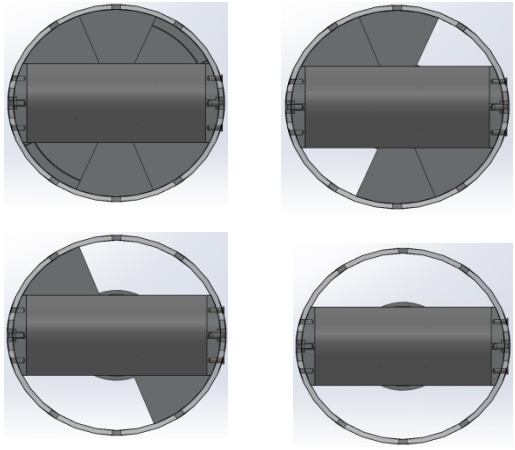


Figure 48 Turning the blade to change the flow area

Say if the number 3 alternative is chosen, it contains a servomotor that provides rotational motion to turn the blade. By turning the blade, it can change the flow area from which the fluid can pass. Figure 48 illustrates this concept. To achieve this functionality, a servomotor is required. It is placed inside of the airfoil. The airfoil makes the housing structure streamlined, otherwise with a blunt body it might incur some oscillation due to wake. It provides housing for the control system other than the servomotor. So, this device serves two main functions, 1. flow control. 2. provide housing for the control system.

7.4 Module-level functional feature implementation considerations

A module-leve functional feature concerns with one or more module-leve functions of a product. A module-leve function is a function that can be fulfilled by a module as the result of functional decomposition from a product-level function. Of course, a module consists of multiple parts and/or other modules, which means that, one needs to further decompose them into part-level functional features. Basically, a module-leve function supports higher level product-level function and can be broken down unto lower level functions.

Usually, a module needs to fulfill one or more functions. It means that multiple functional features could be mapped to one module, which is illustrated in Figure 49, where FF_i means the i^{th} functional feature. Also, other than fulfilling some major functions, a module needs to satisfy some auxiliary functions for the same reason of a part.

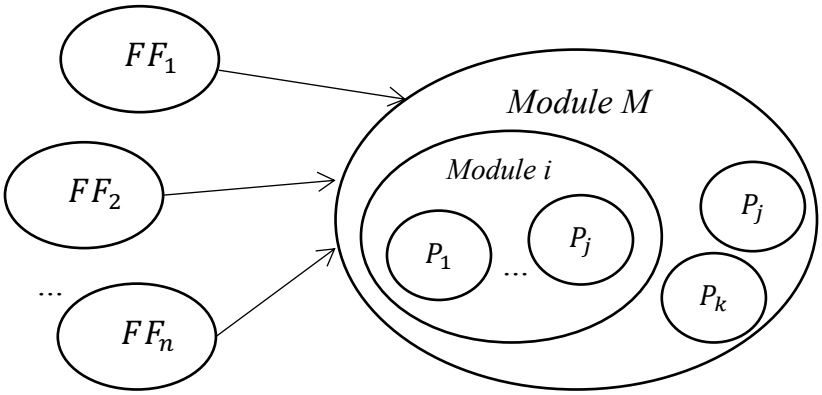


Figure 49 The module-leve functional features schematic

Note that a module needs to be assembled into the product. To do so, there must have some interfaces in the module and other components of the product. If those interfaces are well designed, plug-and-play can be achieved. These interfaces serve the connection function and could be made into functional features.

The key difference between module-level and part-level functional feature is that in the module-leve functional feature the working mechanism is an aggregation of multiple parts. Abstract geometry features for the functional feature of a module might be mapped to one or more parts within the module.

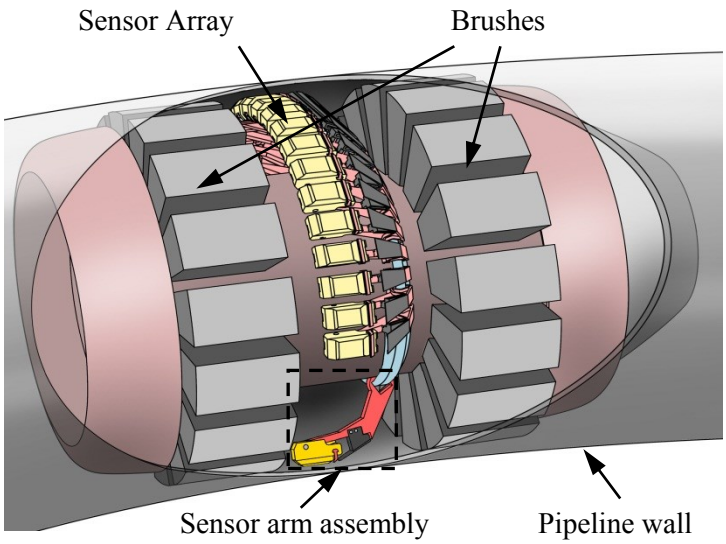


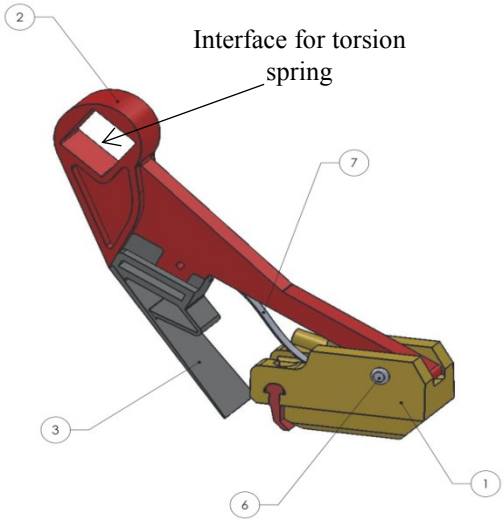
Figure 50 A schematic of a working condition of the sensor arm assembly (adapted from[6])

Let's further consider the example of pipe line inspection gauge. There is a type of intelligent pipeline inspection gauges (pigs) use magnetic flux leakage (MFL) or ultrasonic to inspect the pipeline defects, for example, leaks, corrosions flaws, and wall thickness changes. The inner wall diameter might not always be constant due to abnormalities like corrosions, rust, and scale. A mechanism is needed to ensure that those sensors are always attached to the wall in the case of radius deflection. Sensor arms are parts of the mechanism. A sensor arm subassembly could be seen as a module that satisfies the function of holding the sensor head and connect with the pipeline inspection gauge. A schematic of the working condition for the sensor arm assembly is shown in Figure 50. A complete subassembly of the sensor arm product is given in Figure 51.

[6] Phil Morin, Rick McEwen, Jeremy Hall, Ryan Janzen, Jared Schmidts, Mark Elamatha. Pipeline Inspection Tool Sensor Arm – Conceptual Design Report, 2012, MEC E 460 Capstone Design Project

Based on the above discussions, the sensor arm assembly needs to fulfill the following main functions.

1. rotate when there are obstacles in the contact position of the sensor head (non-rigid connection)
2. hold sensor head
3. keep the sensor head straight without vertical deflection when the sensor arm rotates



ITEM NO.	PART NUMBER
1	Sensor head assembly
2	Arm
3	Bumper
4	Bearing
5	Female Threaded Pin
6	Male Threaded Pin
7	Leaf Spring

Figure 51 An example sensor arm subassembly module[7]

In this case, it is assumed that a square hole is enough as an interface for the sensor arm, where a torsion spring will be placed in between. It designed to be compatible with existing design. So, this square would be an interface made from the sensor arm for this module. Since the sensor arm also needs to connect with the sensor head, it should have an interface to fulfill this connection function.

[7] Phil Morin, Rick McEwen, Jeremy Hall, Ryan Janzen, Jared Schmidts, Mark Elamatha. Pipeline Inspection Tool Sensor Arm – Conceptual Design Report, 2012, MEC E 460 Capstone Design Project

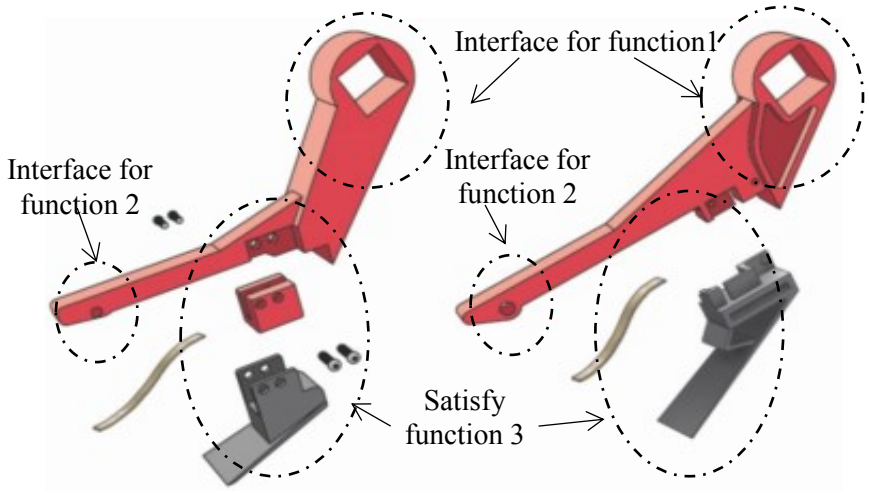


Figure 52 Two alternative modules to satisfy the same functions[7]

Of course there are more than one alternative design to satisfy the above mentioned functions. Figure 52 gives two alternative modules or subassemblies that fulfill the same set of functions listed above. They have same interfaces for them to be assembled to the pipeline inspection gauge product and hold the sensor head. Note that the differences lie in their inner module connection to assemble the bumper onto the sensor arm. The following discussion will be based on the second alternative design.

The effects of satisfying the above listing functions are shown in Figure 53, where a case of changing radius of the pipeline is considered. One of the key of the design is to use an easily deformable bumper, which helps to push the sensor head to rotate. Together with the rotation of the sensor arm itself, it helps to ensure that the sensor head is closely in contact with the pipeline wall. Figure 53 also highlights the key geometry for the bumper to fulfill the function 3, which is likely to be the first part in contact with the bump or feel the change in the radius.

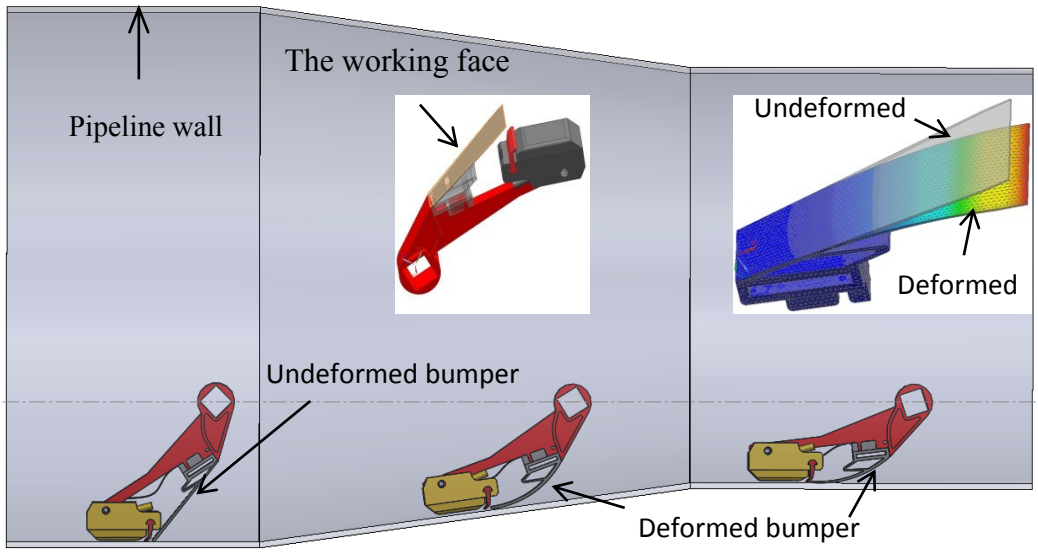


Figure 53 A schematic of different working conditions of the sensor arm module with respect to the changing radius of the pipeline

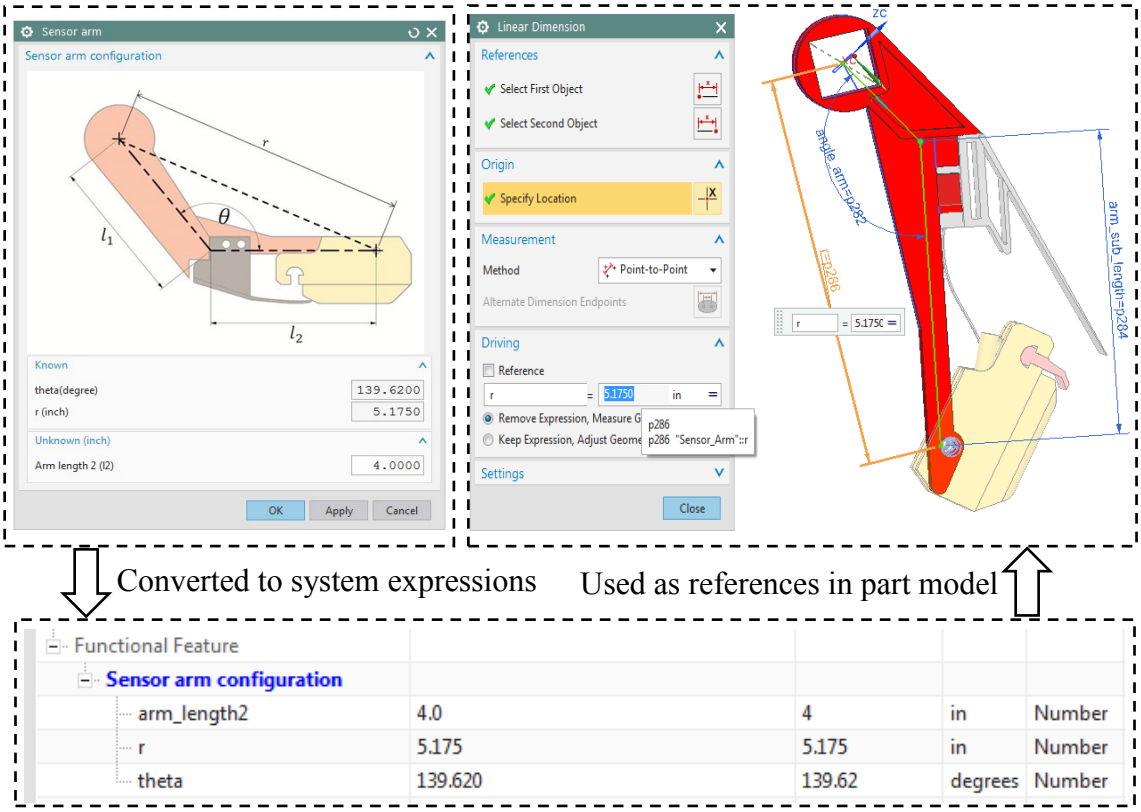


Figure 54 An example application for a module-level functional feature

In terms of incorporating functional feature into the design process of this module-leve product, Figure 54 gives an example of linking module-leve functional feature parameters into the construction of a lower level part by using inter-part expression. A GUI has been constructed to fetch parameters input for a module-leve functional feature. The parameters are then converted into system expressions, which are further referenced by expressions from part modeling. Note that this functional feature can not only fetch user input to create desirable system expressions but also be used to modify those expressions. Since the CAD models can be controlled with expressions, they can be updated with the alterations in the functional feature parameters. Next chapter will discuss the implementation method in more detail.

7.5 Part-level functional feature implementation considerations

A part-level functional feature concerns with one or more part-level functions. A part-level function is a function that can be fulfilled by a part, or some geometry or a part. Part-level functions are the results of functional decomposition of product-level or module-level functions. Thanks to the flexible representation of the abstract geometry feature, in the part-level it can be the, for example, working face, or skeleton. This abstract geometry feature stays within the part, although it can communicate with other abstract geometry feature of others. The parameters might include the geometric parameters of the abstract geometry feature or some functional parameters.

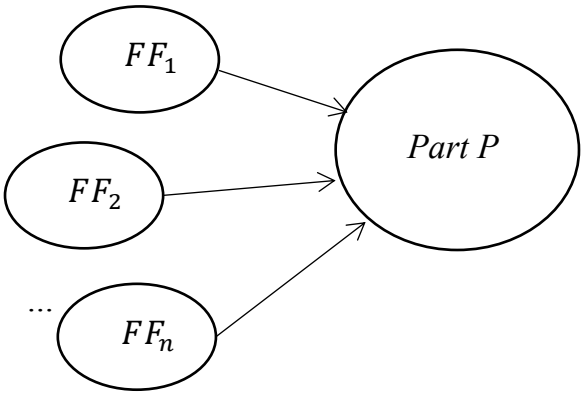


Figure 55 The part-level functional features schematic

When it is the case where a part needs to fulfill multiple functions, multiple functional features exist for this part. Here it might involve more constraints to position different abstract geometry features of different functional features (See Figure 55). The key here is that abstract geometry features for each functional feature are mapped to the geometries of the part.

In the part level, other than some part-level main functions, there exist auxiliary functions as well. For example, since a part need to be assembled to a bigger (sub-) assembly, it needs some connecting interfaces, which could be modeled as functional features, too. For another example, in the process of fulfilling the main function, it might incur some other auxiliary functions that are not available in the higher-level function structure, e.g., module level or product level.

The example of a piston shown in the previous chapter demonstrates part-level functions. Let's revisit it in Figure 56. It could be seen that part-level functions are usually not stand alone. They come from higher level functions. Part-level functions are fulfilled by its key geometries.

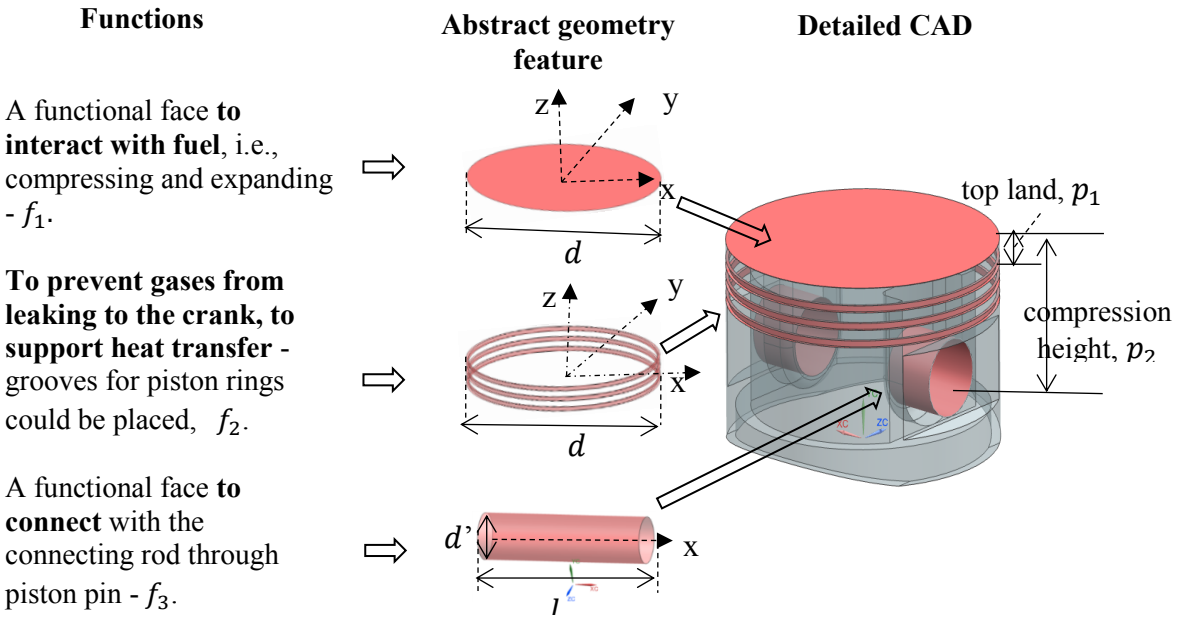


Figure 56 A part-level functions example from the previous chapter

7.6 Chapter summary

This chapter gives a possible semantic definition of a generic function and discusses the function aspect of applying functional features in different levels of product design, including part-level, module level, and product level. By breaking them down into different levels allows reasoning at different levels of abstraction. The geometry aspect of the functional feature is deliberately not discussed in detail in this chapter because it has been discussed extensively in the previous chapter. More comprehensive cases are to be presented in next chapter.

8 Case Studies for Functional Feature Modeling

8.1 Chapter introduction

This Chapter presents three case studies, the design of a slotted liner product, road crossing, and engine subassembly, focusing on different aspects of the design. The first case study addresses the application of functional feature in the part modeling level. The last two extend the application to the assembly design. The road crossing case is mainly used to show that the assembly configurations could be captured with abstract geometry features. The engine subassembly incorporate more programming and GUI design based on the proposed functional feature modeling.

8.2 Slotted liner case study

The design of a slotted liner is chosen to be the first case study in this chapter. It shows how functions of the product are closely related to its geometries and it demonstrates that the concept of the functional feature is applicable to encapsulate and represent such relations into a manageable knowledge unit and could be embedded into a CAD model.

Slotted liners have been used in horizontal wellbores to maintain the borehole integrity, prevent formation collapse, and provide sand control in, for example, Steam-Assisted Gravity Drainage (SAGD), which are commonly applied in heavy oil recovery in western Canadian. They are manufactured from tubes by saw-cutting slots with different configurations, e.g., slot shape, slot width, and slot pattern, etc. The geometry of slotted liners is simple with only dozens of geometrically related parameters. However, it makes little sense when staring at the numbers

used in the CAD modeling without knowing their meanings, interactions, and rationales. To make more sense out of the numbers functional considerations need to be involved.

Two functions will be examined in the case study. It can be seen that the purposes of a slotted liner are, on the one hand, to allow flow (oil and water) to pass into the tube for production, and on the other hand, to prevent sand particles from entering the tube. In terms of the flow of energy, material and/or information transformation, it is clear that materials (oil, water/steam, and sand particles) will be passing in and out of the slotted liner with energies (e.g., the steam carries heat out to warm up the reservoir and reduce the viscosity of heavy crude oil). If the functions are studied in to do forms, it is natural to put the functions of slotted liner into "to control the extent of sand particle from passing into the tube" and "to control the amount of flow (oil and water/steam) in and out of the tube (depending on whether it is used in injector or producer of SAGD). Henceforth these two functions will be abbreviated as sand control and flow control in the case study.

Figure 57 presents an overview of the basic elements involved in functional features in the model of slotted liner. Two functional features will be considered in the current study, i.e., sand control functional feature and flow control functional feature, to incorporate the sand control and flow control functions of the slotted liner into the CAD model. Those functionalities are closely related to the geometry of the slotted liner, as well as the involved engineering physics with constraints.

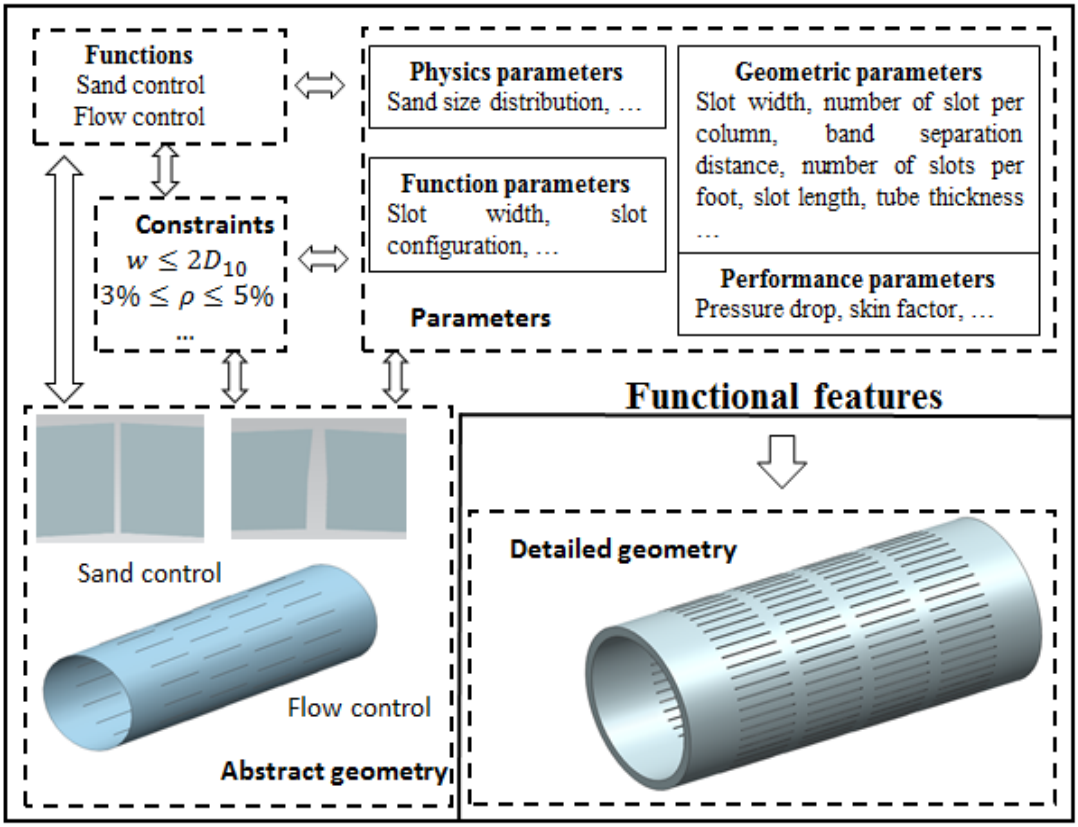


Figure 57 An overview of functional features in slotted liner design

8.2.1 Sand control functional feature

The functionality of sand control is closely related to the shape and dimensions of each slot. In terms of the shape of the slot, straight slot and keystone slot are normally used (Figure 58). In a straight slot, its widths are the same in both the inner and outer wall of the pipe; in a keystone slot, it is wider in the interior wall and narrower in the liner, which promotes self-cleaning and hinders plugging. The width of the slot w, which is critical to sand control efficiency, should not be larger than twice the D_{10} sand grain diameter. The D_{10} parameter, which is non-geometric, represents the particle diameter corresponding to the 10% cumulative undersize particle size distribution, indicating that only 10% of the sand sample quality can pass through the slot (Wan

2011). The length of the slot is usually determined by the width of the slot[8].

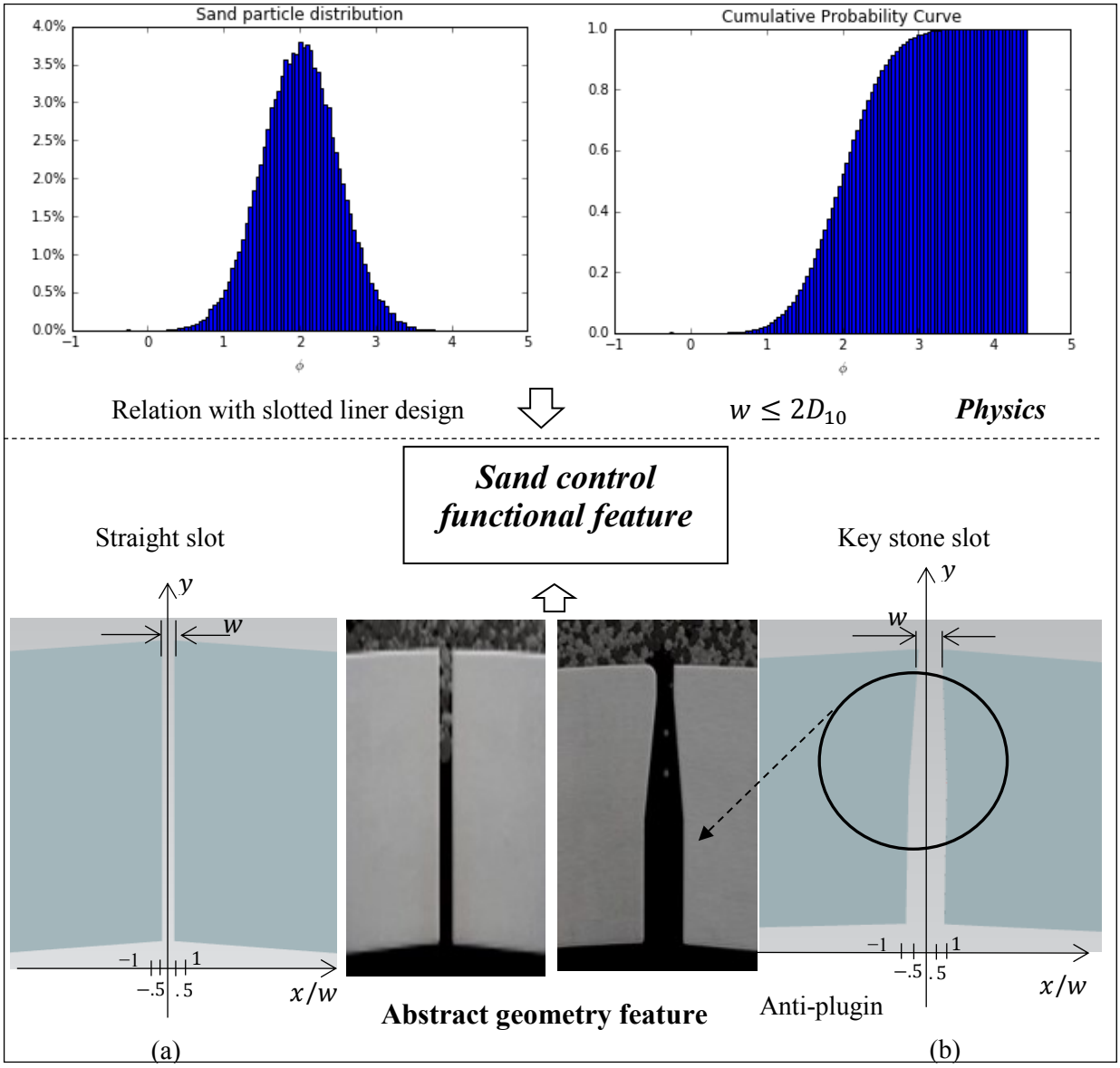


Figure 58 The sand control functional feature bridges information from sand particle distribution to slot width

Figure 58 gives an example of sand particle size distribution and sand control functional feature relates that knowledge to the design decision made in determining the geometric parameter, e.g., slot width, of the slotted liner. Note that ϕ is a measure of particle size and is

[8] Pacific, Perforating Inc., Tubular Slotting. http://www.pacificperforating.com/tubular-slotting.php#2 (Accessed July 22, 2015)

defined as the negative log base 2 of the diameter d in mm (Blott and Pye 2001; Krumbein 1934).

The abstract geometry feature of the sand control functional feature is the profile of every single slot, which might be of shape straight or keystone. The physics feature involved is the equation specifying the relationship between sand particle size distribution and slot width. The parameters involved in this specific functional feature include the geometric related parameters for the slot, i.e., slot width and length, as well as the links to the sand particle database for the specific reservoir. The constraints dictate the relation between the slot width and the diameter of medium-sized sand particles. By abstracting the sand control functional feature engineers can change the functional parameters to suit the upstream requirements specified by the characteristics of the reservoir and the CAD model of the product would update automatically.

8.2.2 Flow control functional feature

Flow control functional feature closely relates the physics phenomenon with the product geometry. The design of slotted liner is a result of interacting factors like reservoir permeability, permeability anisotropy, fluid properties, formation damage effects, and rock mechanical characteristics, etc. (Furui et al. 2007). It is crucial to understand the flow field in a single slot (Ansari et al. 2015) as well as the combinatorial effect of the multiple slots (Kaiser et al. 2002) distributed in the slotted liner, based on which an optimum design could be achieved. Under the working condition, slot plugging, in which sand particles bridge the slots, is likely to occur. It reduces the slotted liner to become an extension of the reservoir material and cause the flow through the slot to turn into Darcy flow instead of open channel flow (Furui et al. 2005; Kaiser et al. 2002). Under this condition, it is safe to argue that the configurations or slot patterns have

great influence on the fluid flow, both inflow and outflow, which should be examined with scrutiny. Figure 59 shows the overview of the flow control functional feature.

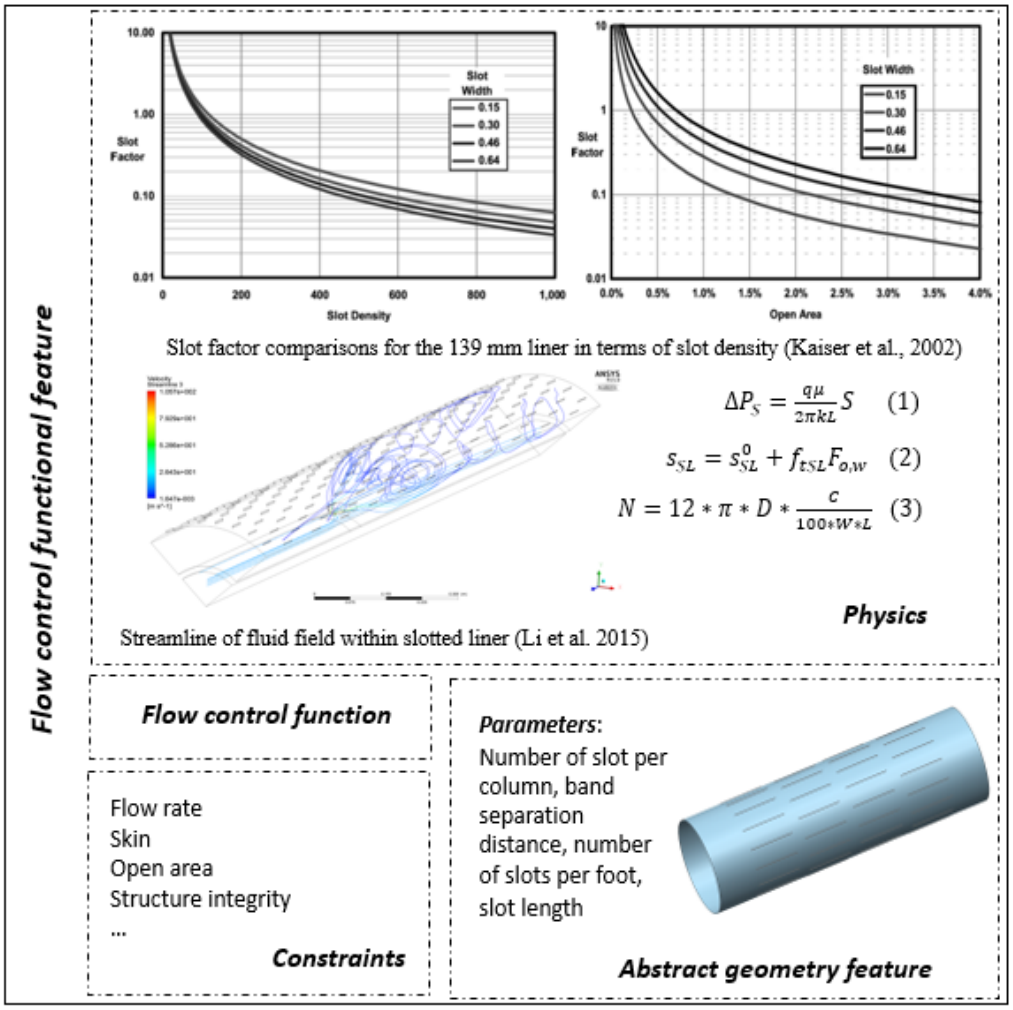


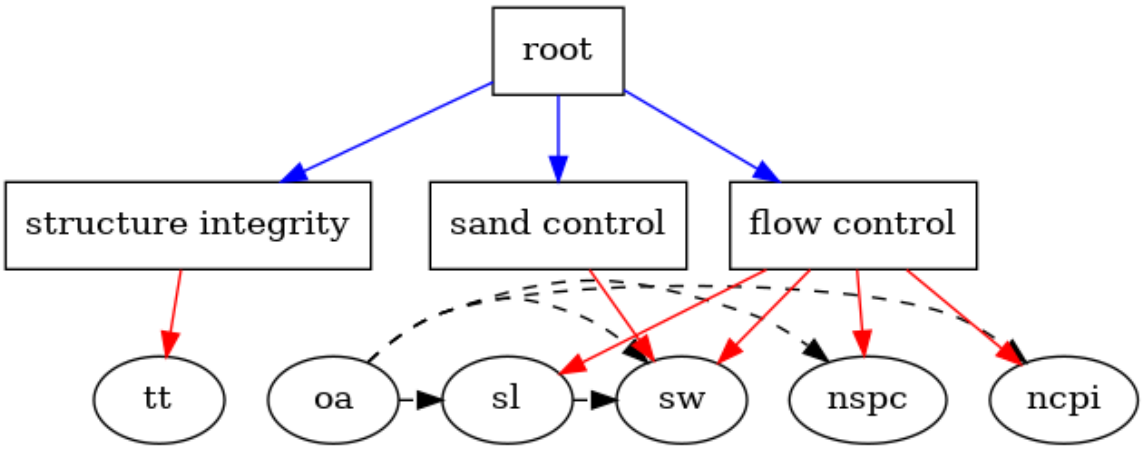
Figure 59 An overview of the flow control functional feature

The cumulative effect of multiple slots is characterized by the rate independent skin or slot factor, which is determined by elements like open area, slot density, slots per column (SPC), and is related to the pressure drop (Furui et al. 2005), as is shown in equation 1 of Figure 59. A skin model for slotted liner completion is given in equation 2, details could be found in (Furui et al. 2005). Kaiser et al. (2002) show an example of the effect of slot density or open area in the slot factor (presented in Figure 59). The targeted open area is often set upfront based on the required

production rate. If the slot width, which is controlled by sand control functional feature, is small, slot density has to be high to maintain the specified open area. The value of N, number of slots per foot, is related to liner diameter, open area, slot width, and slot length (see equation 3 in Figure 59).

8.2.3 Mapping of engineering parameters to geometric parameters

Figure 60 presents the mapping of functions and some key parameters of the slotted liner design. Note that the root function is a dummy function for the product at hand and it is broken down into three smaller functions. Two functions are of interest here and have been discussed in the above section. Functions are shown in the blocks and parameters in the ellipses. The mappings from function to parameters are shown in red and between parameters in dotted black. Note that in this specific example, flow control function and sand control function share a parameter "sw", i.e., if the value of "sw" changes it will affect both functions. This graph can be expanded if we consider more functions and parameters of the design.



Nomenclature:
oa: open area, sl: slot length, sw: slot width, nspc: number of slots per column, ncpi: number of columns of inch, tt: tube thickness

Figure 60 The mapping among key functions and parameters of the slotted liner design

Engineering parameters are those parameters engineers care about in the design, which might or might not be directly geometric related. The mapping of engineering parameters to the parameters used in the construction of CAD model requires some engineering scrutiny. Such mappings, constructed based on mathematical equations, e.g., equation 1 -3 in Figure 59, need to be embedded into functional features to achieve the associativity between the designs in general and CAD modeling.

Table 6 Slotted liner functional features and related parameters

<table>
<tr><td rowspan="5">Slotted liner</td><td colspan="3" rowspan="2">Functional features</td><td colspan="2">Parameters</td></tr>
<tr><td>Non-geometric</td><td>Geometric</td></tr>
<tr><td rowspan="2">Sand control</td><td rowspan="2">Slot shape</td><td>Straight</td><td rowspan="2">Sand particle size distribution</td><td>Slot width</td></tr>
<tr><td>Key stone</td><td>O.D. slot width</td></tr>
<tr><td>Flow control</td><td>Slot configuration</td><td>In-line
Staggered</td><td>Slot density, skin, pressure drop …</td><td>Number of slot per column, band separation distance, number of slots per foot, slot length</td></tr>
</table>

With the above discussion, it could be found that the slot width is closely related to the sand control function, and the opening area related parameters are pertinent to the flow control function (Table 6). The changes of those parameters directly alter the CAD model and provide a minimum “interface” to interact with the CAD geometry. Moreover, parameters associated with

geometry and parameters associated with, e.g., function or physics features, are also dependent on each other. Those parameters are embedded into functional features to provide an enriched engineering semantics. The expected result is that whatever reasonable requirements come into the system, the model could always be updated properly and accordingly without the tedious re-modeling of the product by altering either the geometry related parameters or engineering parameters (mostly non-geometric).

8.2.4 A design prototype

A prototype result is shown in Figure 61, which also shows the underline feature dependency graph for the model construction. It could be seen that only functional interfaces are exposed to the users and unnecessary construction details are hidden. Functional features are applied in here to model the sand control and flow related functions of the slotted liner such that engineers or end users are presented with the functional product information instead of tedious and error-prone modeling operations to assist design communication and speed up design changes since change propagation are handled internally within functional features without user interferences.

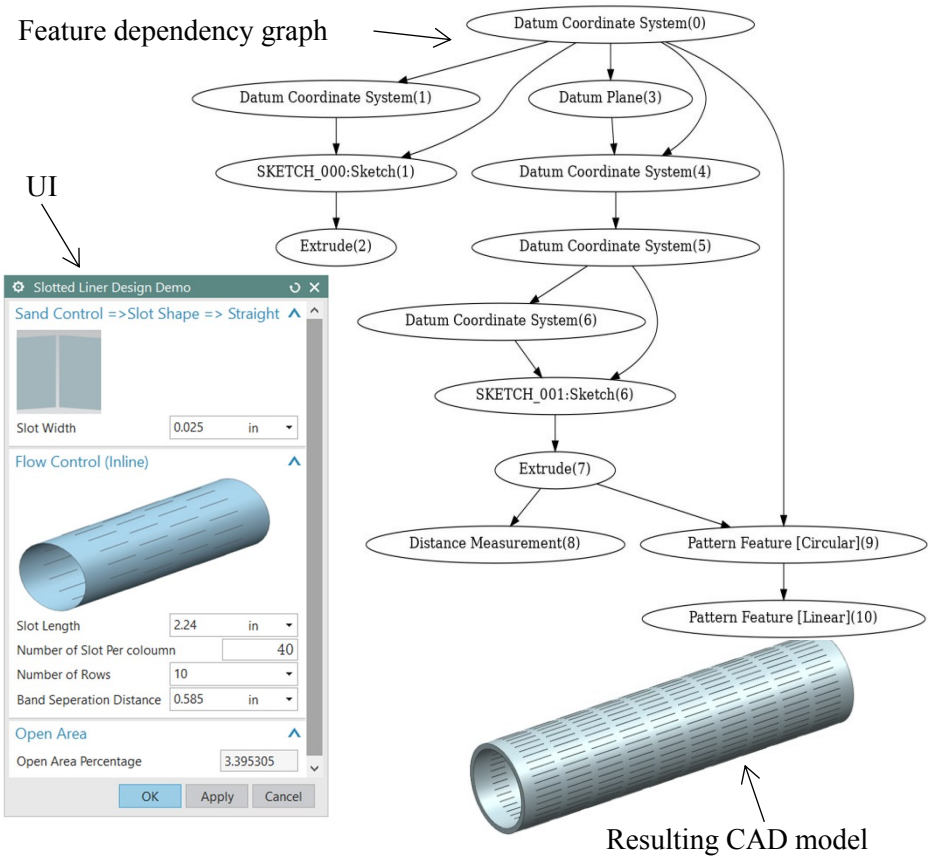


Figure 61 A design prototype of the straight inline slotted liner

It is admitted that functional features identified in the current case study are not complete. For example, another functional feature of interest might be related to the structural integrity of the slotted liner. One could also consider analyzing the manufacturing related functional feature as the shape of the slot is determined by the applied manufacturing method (Figure 62), i.e., instead of a rectangular shape section, each slot should have a have a curvature that is constrained by the cut of sawing blade. The main purpose of the current case study is not to provide a complete list of the product functional features, but to give examples to demonstrate the potential of functional features in linking the functions and geometries of a product together and assisting a function driven design process.

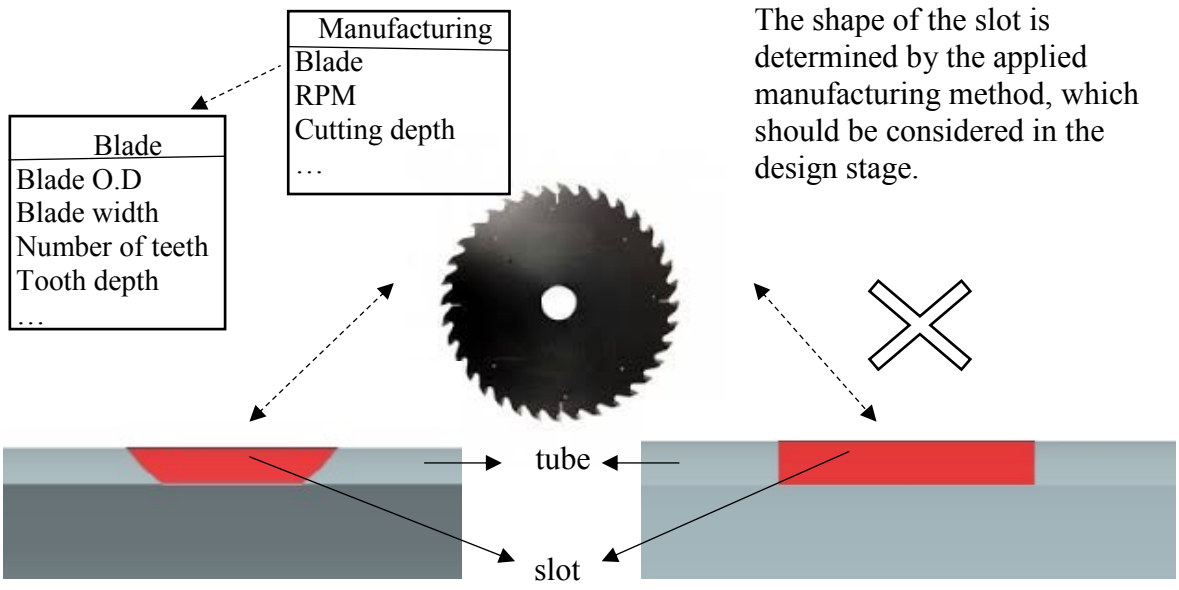


Figure 62 Manufacturing related functional feature that correlates the shape of the slot with a manufacturing method

8.2.5 Coping with design changes

With functional feature support, design changes could be triggered from the functional perspective without dealing with intricate modeling operations. Basically the sand control and flow control functional features involved in the design of slotted liner encapsulate two possible functional changes in the model, the functionalities of which have been embedded in the CAD model such that the model is agile and responsive to functional changes. Figure 63 and Figure 64 depict the functional change scenarios. For example, if the slotted liners are to be used in reservoirs with different granular sand distributions, by changing the design requirements, the functional performance of the design needs to be updated automatically by adjusting the design parameters. Likewise, the slot density of the design is responsive to the functional requirements of flow control. Instead of manipulating feature operations in feature tree in an unintuitive manner, engineers can interact with the functional interfaces directly.

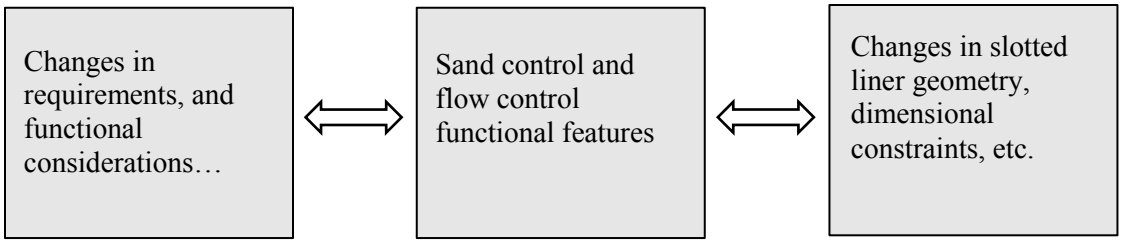


Figure 63 Functional features handle the possible functional changes

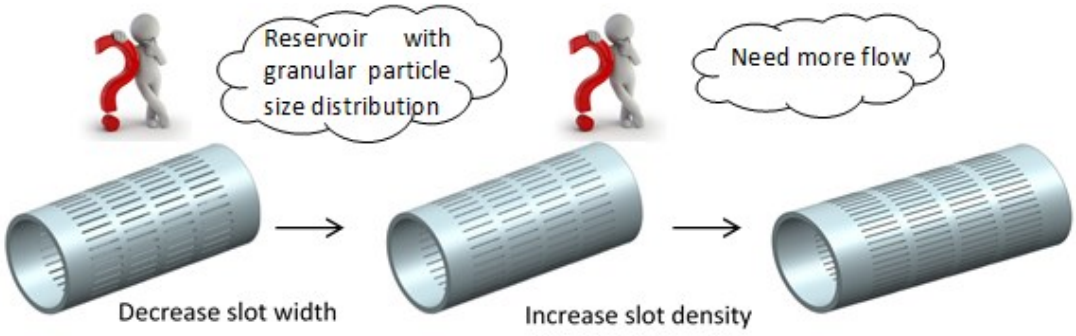


Figure 64 Examples of functional change scenarios in the design of slotted liner

8.3 Road crossing case study

This section presents a case study involving assembly structures to demonstrate the feasibility and effectiveness of functional feature modeling in the assembly design context. Function-wise, road crossing, or road ramp, is used as a temporary pipeline to transport fluid crossing roads without blocking traffic. To satisfy the main functions, major structures like flanges and HSS tubes are used.

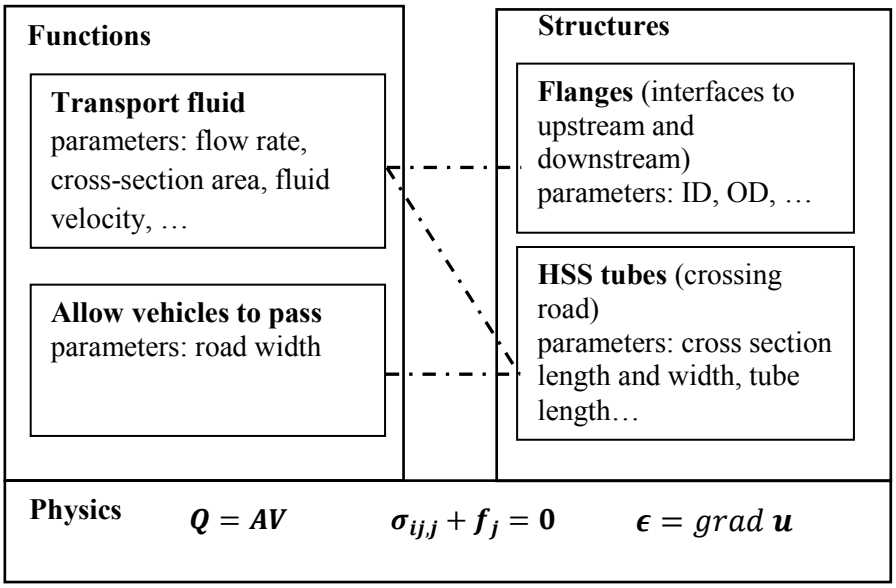


Figure 65 Main functions, structures, and physics of a road crossing

The mappings among the main functions and structures are shown in Figure 65, together with the major physics considerations in terms of the flow rate and stress. Note that the flanges provide interfaces to upstream and downstream flows. The HSS tubes are to be placed on the road to allow vehicles passing by. There are also some supporting structures to satisfy auxiliary functions. For example, since HSSs and Flanges are different in shape an intermediate structure is required to connect them together. Figure 66 shows the component list for the road crossing structure. They are indispensable to form the road crossing unit to fulfill the functions specified above.

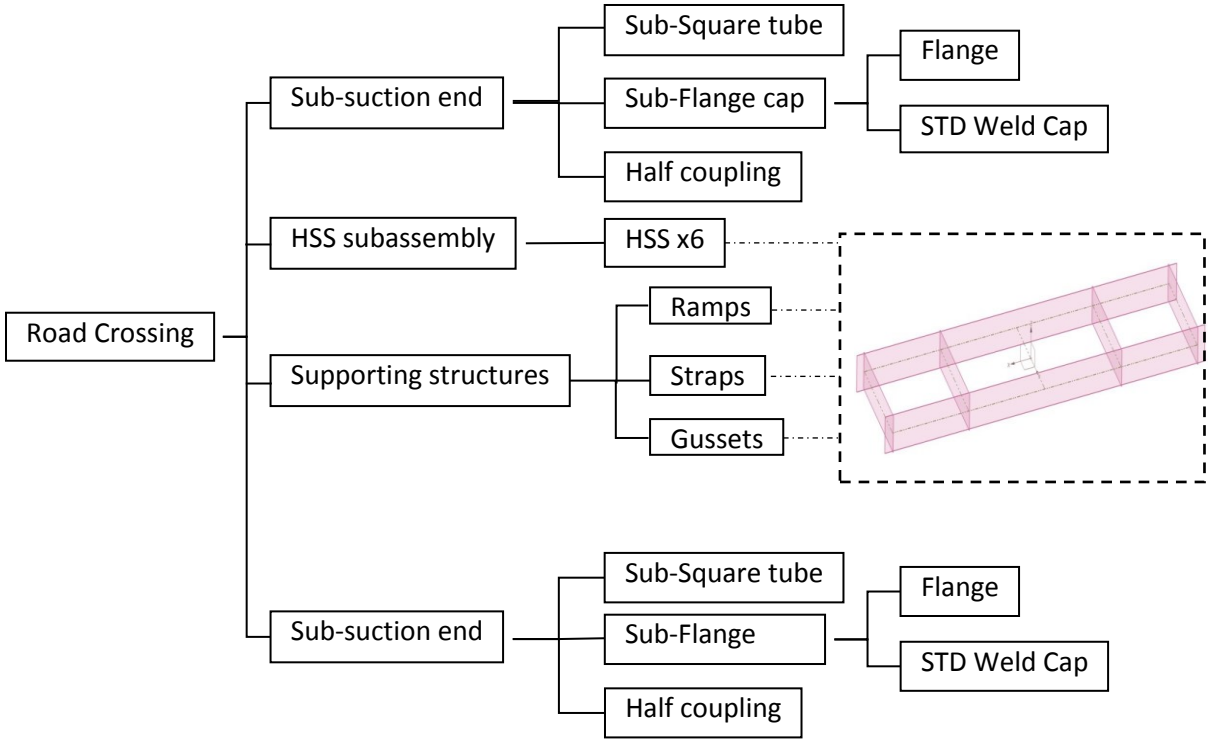


Figure 66 The road crossing assembly structure

Figure 67 shows the mapping among key functions, parameters, and parts of the design. Functions are shown in boxes, where blue arrows denote function decomposition with a dummy root function. Parameters are shown in ellipses and parameter dependencies are indicated in dotted black arrow. Parts are shown in hexagons where dashed lines are used to show assembly relations. Functions, parameters, and parts are grouped together. The mappings from functions to parameters are shown in red arrows, from functions to parts in sienna arrows, and from parameters to parts in green arrows. Note that this graph is not meant to be comprehensive but to show the basic concept of mapping among different design elements. For example, in the parts cluster only three key parts are shown. A more comprehensive assembly structure can be seen in Figure 66.

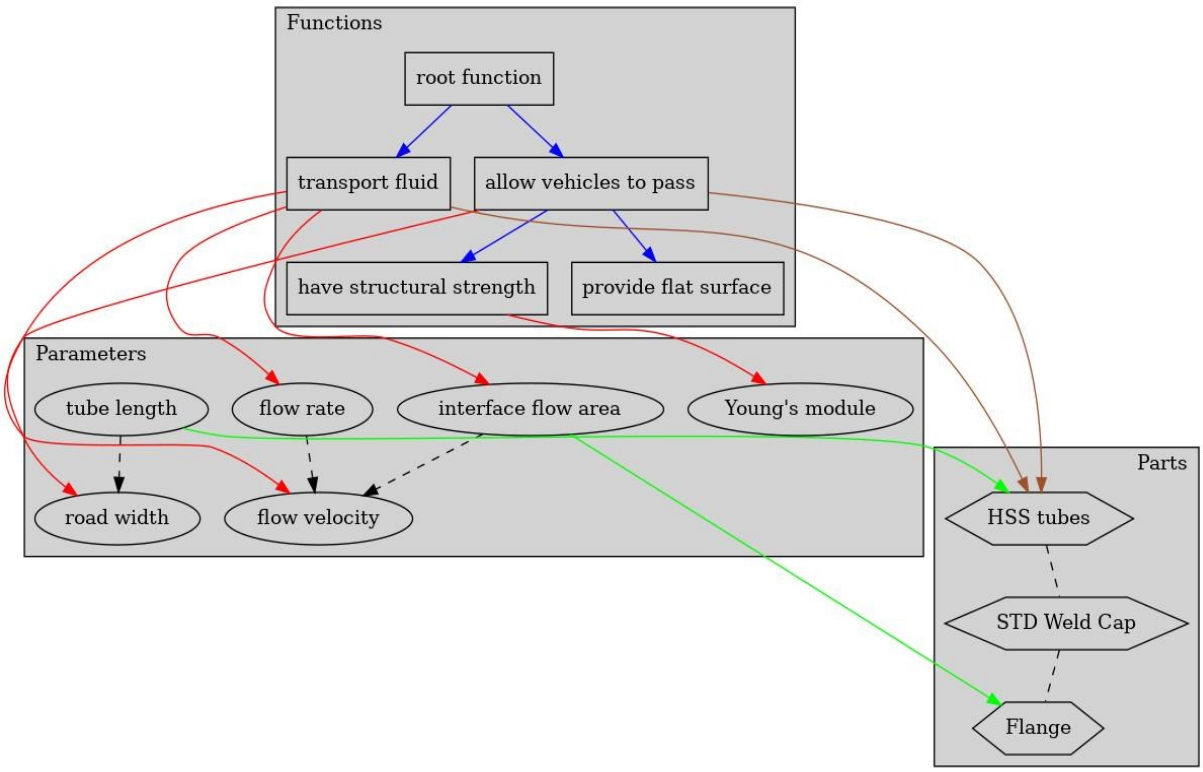


Figure 67 The mapping among key functions, parameters, and parts

Let's examine what contributes to the functionalities provided by the HSS tubes in terms of its abstract geometry features. Figure 68 demonstrates the abstract geometry features with the HSS tubes. For the functionality of transporting fluids, it's clear that the inner volume of the HSS is the fluid domain. The cross-section of the fluid domain provides the area to calculate the flow velocity based on a given flow rate. The functional face that allows vehicles to pass is the upper surface of the HSS tubes, which is highlighted in Figure 68. Of course, the face alone cannot support the weight and impact of vehicles. The HSS tubes and other supporting structures are working together to fulfill the function.

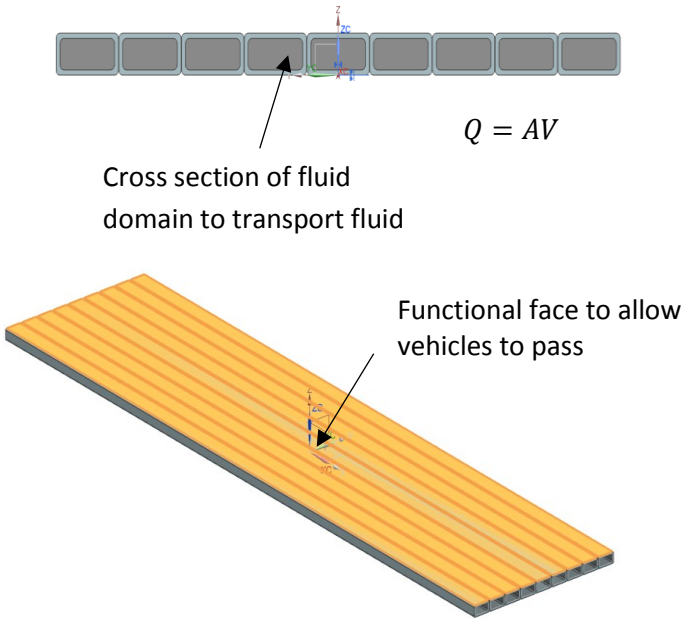


Figure 68 Abstract geometry features in HSS

Other than discussing functional features within a single road crossing, this section also wants to show a configuration function that contributes to the assembly design of the road crossing product family. Since different roads have different widths, the road crossings need to adapt to a variety of different roads. For example, Figure 69 shows two road crossings with two different configurations.

This functional feature, together with the corresponding abstract geometry feature, provides guidance in the assembly modeling of the road crossing. Figure 70 provides an overview of the configuration functional feature. It can be seen that the geometric aspect of the abstract geometry feature is very primitive. It provides the skeleton and characteristics of the product family.

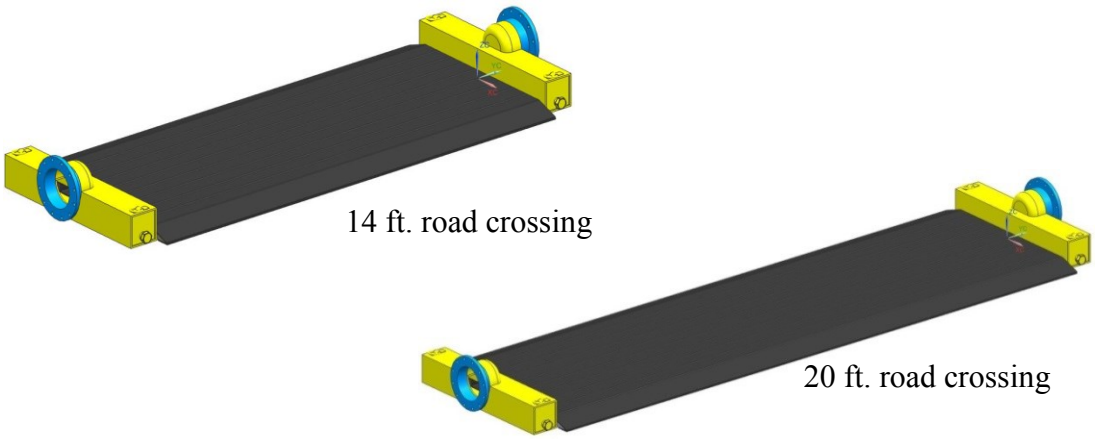


Figure 69 Two examples of road crossing with different configurations

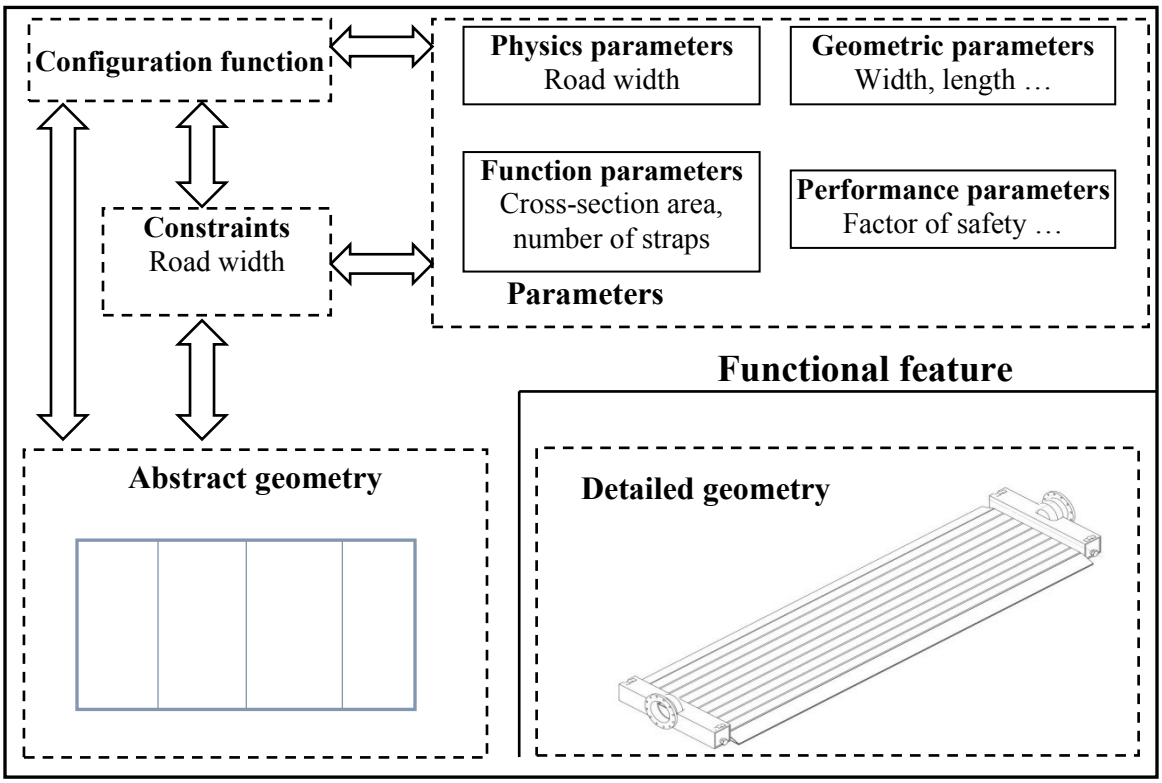


Figure 70 Overview of the configuration functional feature in the current case study

8.3.1 Preliminary analysis

Here we want to build a configuration functional feature that controls the configurations of different products within this product family. The main varieties of the road crossings are the

lengths. The length of the road crossing needs to be in line with the width of the road. At the first glance, it might seem trivial to change the length. However, it is more than that. The length of the road crossing cannot be changed without other modifications to make it strong enough, e.g., with the supporting structures, including straps, ramps, and gussets.

Changed parameters	**Unchanged parameters**
Number of gussets Length of the HSS tubes Number of straps …	Suction end subassembly Cross section of the HSS tube …

Figure 71 Examples of the results of the preliminary analysis

Under this circumstance, there is something unchanged for this configuration. So for the preliminary analysis, we need to identify what components are involved in the design of the road crossing and which will be changed in different configurations and which stay the same. Figure 71 gives a few examples of the result of the analysis. The suction end subassembly stays the same and the type of HSS is unchanged. However, the length of HSS and the number of supporting structures need to be adapted to different design requirement for different road length. Figure 72 shows one of the example models that stay untouched in different length configuration of the road crossing, a suction end subassembly. This could be saved as a template to be reusable.

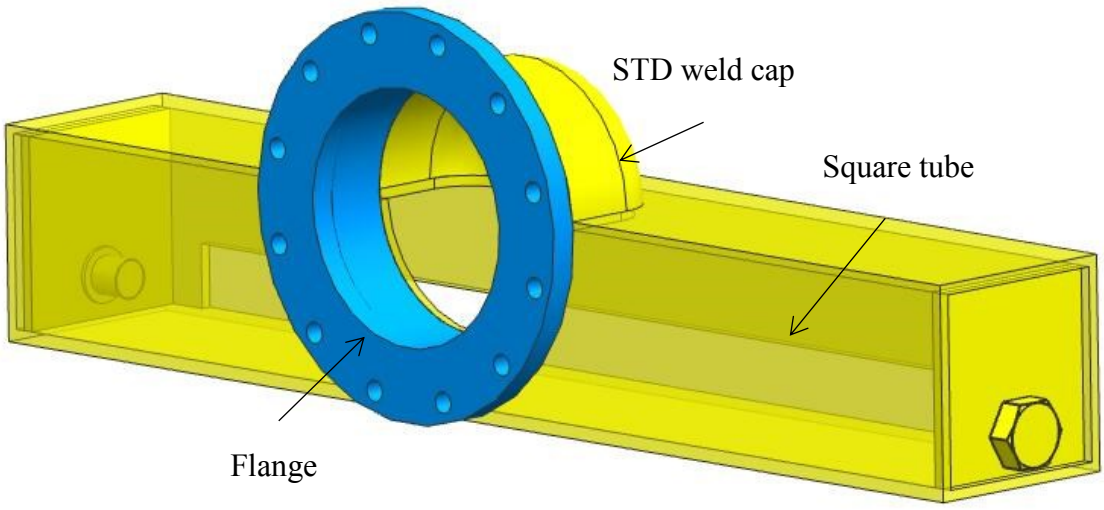


Figure 72 Suction end subassembly that is unchanged across different configuration

Table 7 An example parameterization for the flange

Dimensions of Flange RFWN 12" 150 STD A105N							
Nominal size		**Mating Dimensions**					
DN	inch	Outside Diameter [inch]	Diameter of Bolt Circle[inch]	Diameter of Bolt Hole L [inch]	Bolting Number	Bolting Size [inch]	Flange Thickness C [inch]
300	12	19	17	1	12	7/8	1.25

8.3.2 Parametric modeling of each component and subassembly design

Next step is to model each component parametrically. It is desirable to adopt a parametric approach to make the models well reusable. Note that the reusability does not mean that the dimensions are fixed and could be used without any changes. Rather, it means that the model is easily changeable with alteration of parameters in the model. For example, the length of the HSS tube of the following can be readily modified by a single parameter. It is what we need now

because we are interested in configuring the length of the road crossing. For example, Table 7 shows a key parameterization of a flange.

The same process could be repeated with other components to have all the ingredients required to assemble the product, for example, see Figure 73. The construction of each part is not hard, which could be seen from the resulting feature dependency graph. Figure 74 gives two example feature dependency graphs for the HSS tube and STD Weld Cap. Other than individual component, some subassembly could be preconfigured. For example, the HSS subassembly could be constructed with or without the configuration input because the assembly constraints are independent of them. The only variation is the length of the subassembly, which is controlled by the length of the HSS tube. At this stage, the HSS tube could update its length based on the configuration input.

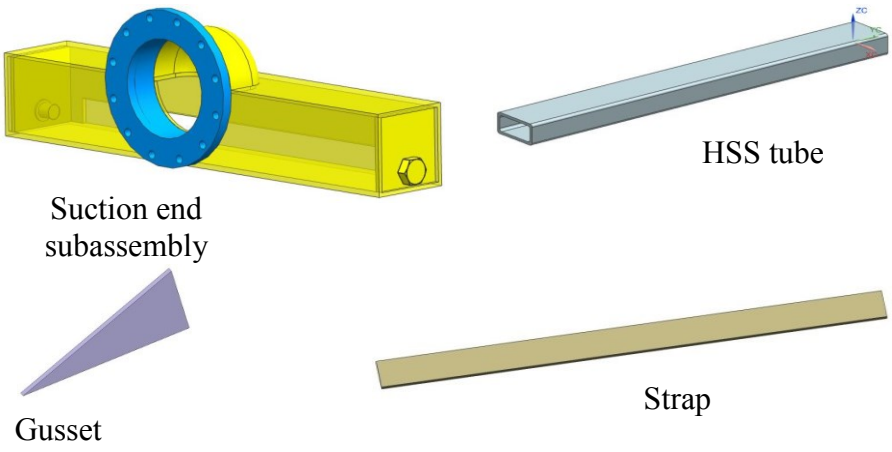
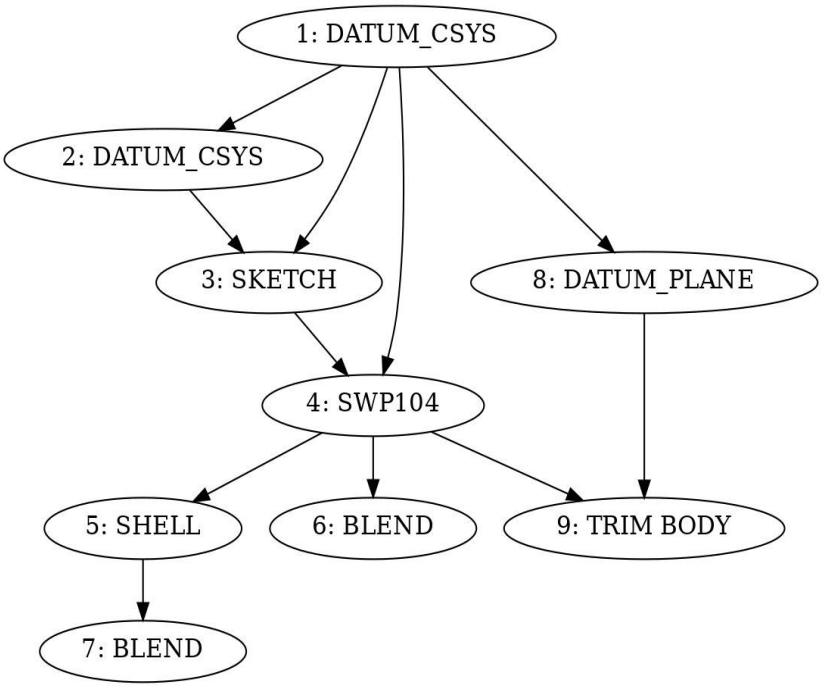
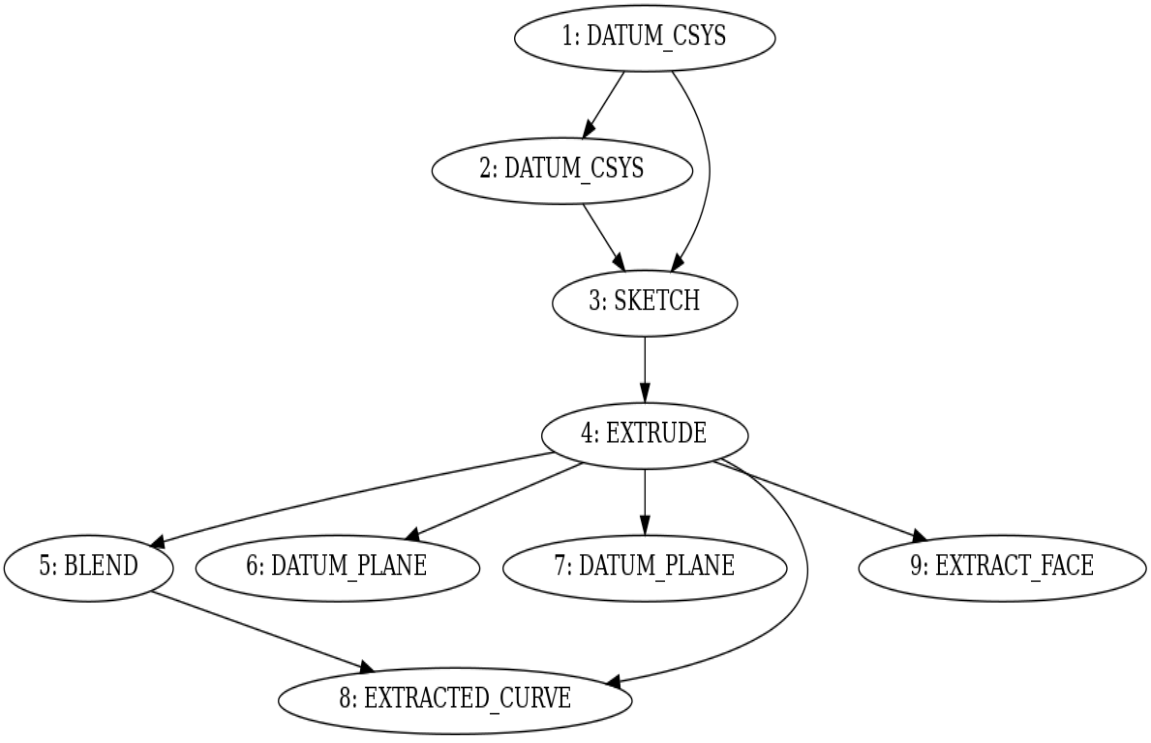


Figure 73 A few examples of reusable components in the road crossing model



(a) The feature dependency graph for the STD Weld Cap



(b) The feature dependency graph for the HSS tube

Figure 74 Feature dependency graphs for two parts

8.3.3 Assembly skeleton design with abstract geometry feature

We start with a skeleton to configure the length of the road crossing according to the need due to the varieties of the road width. This skeleton is the abstract geometry feature (Figure 75). Geometrically speaking, it consists of a few lines that are easy to construct. Parametrically, it has the option to change with the width, the length, and the number of the supporting structures (represented as lines as well). The geometric and parametric information contained in this abstract geometry feature will be used in the assembly process.

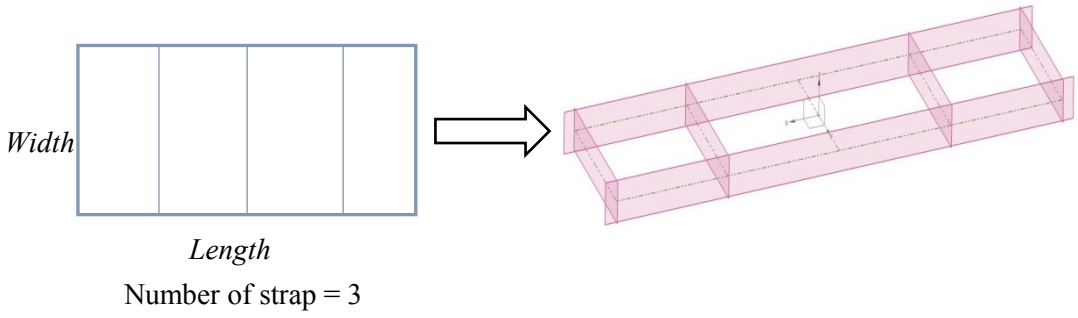


Figure 75 The abstract geometry feature for the current case study

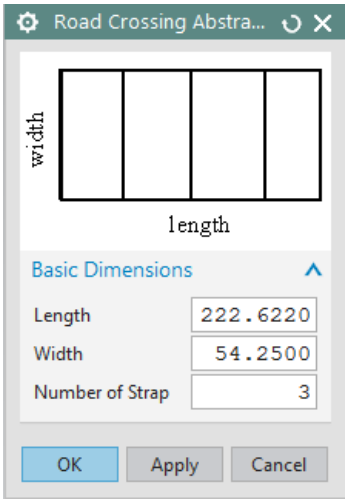


Figure 76 An example GUI to get user configuration input

With the simple sketches, datum planes could be added to the abstract geometry feature to be used as references later in the assembly context. This process, including sketch drawing and

adding datum planes, could be manually or automatically carried out based on the configuration inputs. Here the automatic approach is adopted. A GUI can be created to receive user input for the configuration information, see Figure 76. Note that the width of the road crossing is set to a default value and it is not changeable in this case.

Based on the abstract geometry feature, many other properties could be inferred. For example, the length of the HSS tube and where to place the HSS tubes, the length of the ramp, the number of gusset and strap, etc. The inferred parameters could be stored in the abstract geometry feature for later usages. For example, the number of gussets depends on how long the road crossing is. Figure 77 gives a simple example of how to create inferred expressions by using C++.

```
NXOpen::Expression *expression_interval_gueest, *expression_half_number_gueest;
expression_interval_gueest = workPart->Expressions()->CreateWithUnits("interval_gusset=24", unit);
expression_half_number_gueest = workPart->Expressions()->CreateExpression("Integer",
    "half_number_of_gusset=floor(0.5*length/interval_gusset)+1");
```

Figure 77 Create expressions with NX Open C++

8.3.4 Assembly design based on the abstract geometry feature

The configuration functional feature provides supports during the assembly design. The abstract geometry with datum delivers multiple references to position different components. According to the design requirements, one of the gussets is attached to the suction end subassembly and others are placed 24 inches apart. So the number of gussets could be calculated based on the configuration parameters, which is stored in the configuration functional feature. Figure 78 illustrates this point. During the assembly stage, gusset could be linearly patterned. This operation requires the number of the guest instances, which is set according to the number calculated above.

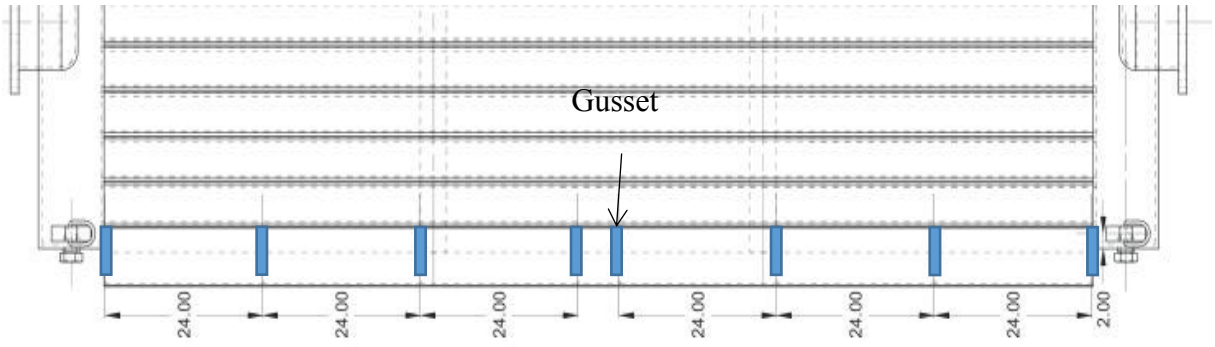


Figure 78 Illustration of the position of gussets based on configuration inputs

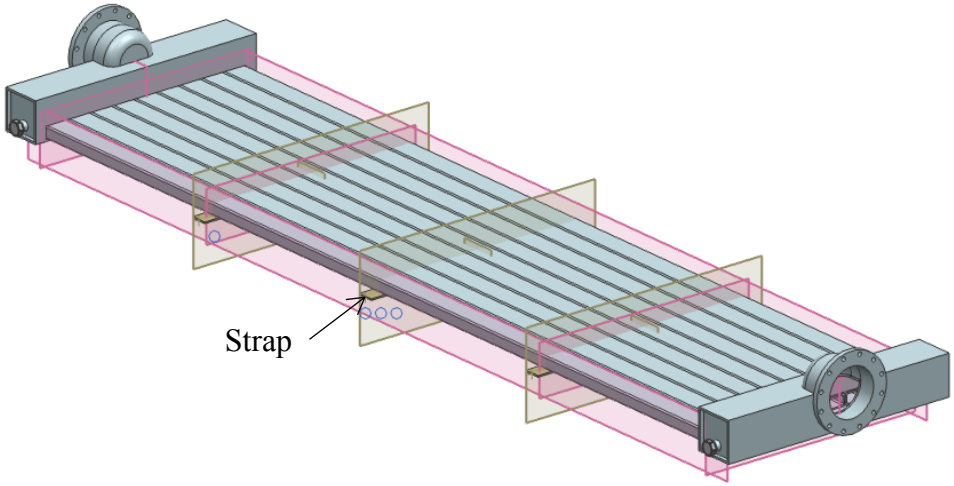


Figure 79 The positions of straps are based on datum in abstract geometry feature

Similarly, the position of straps could also be determined by the abstract geometry feature, as is shown in Figure 79. As a matter of fact, not only the positions but also the number of straps can be dynamically adapted to the abstract geometry feature.

8.3.5 Results and discussions

Figure 80 shows an example workflow with the abstract geometry feature support in the road crossing case study. For this design, engineers don't have to repetitively construct the abstract geometry feature for different configurations. Rather, the abstract geometry feature could be automatically generated based on simple inputs from users. This abstract geometry feature

controls the dimensions of other component and subassemblies. Further, datum could be produced based on the abstract geometry feature and they provide references from which assembly structure could be constructed. Even the assembly process could be automated. However, since here it is just meant to show how functional feature with abstract geometry feature could be used to assist assembly design process, we will not show how to automate the assembly process.

The simple GUI is designed to get the configuration parameters from a user. These configuration parameters are used to construct the abstract geometry feature. However, if we stop here, the model would not be responsive to design changes. In order to address this issue, parameterization in the CAD system is necessary. Parameterization is more than just using parameters to define and change the design features. It can be used to drive the whole design. This is possible because each expression, which is the technical implementation of parameterization in Siemens NX, remembers its owning and using features, as well as the referencing expressions. When one expression is changed, the system can check and update whatever features and expressions that are involved. This type of paradigm is well applicable to other product families like trailers.

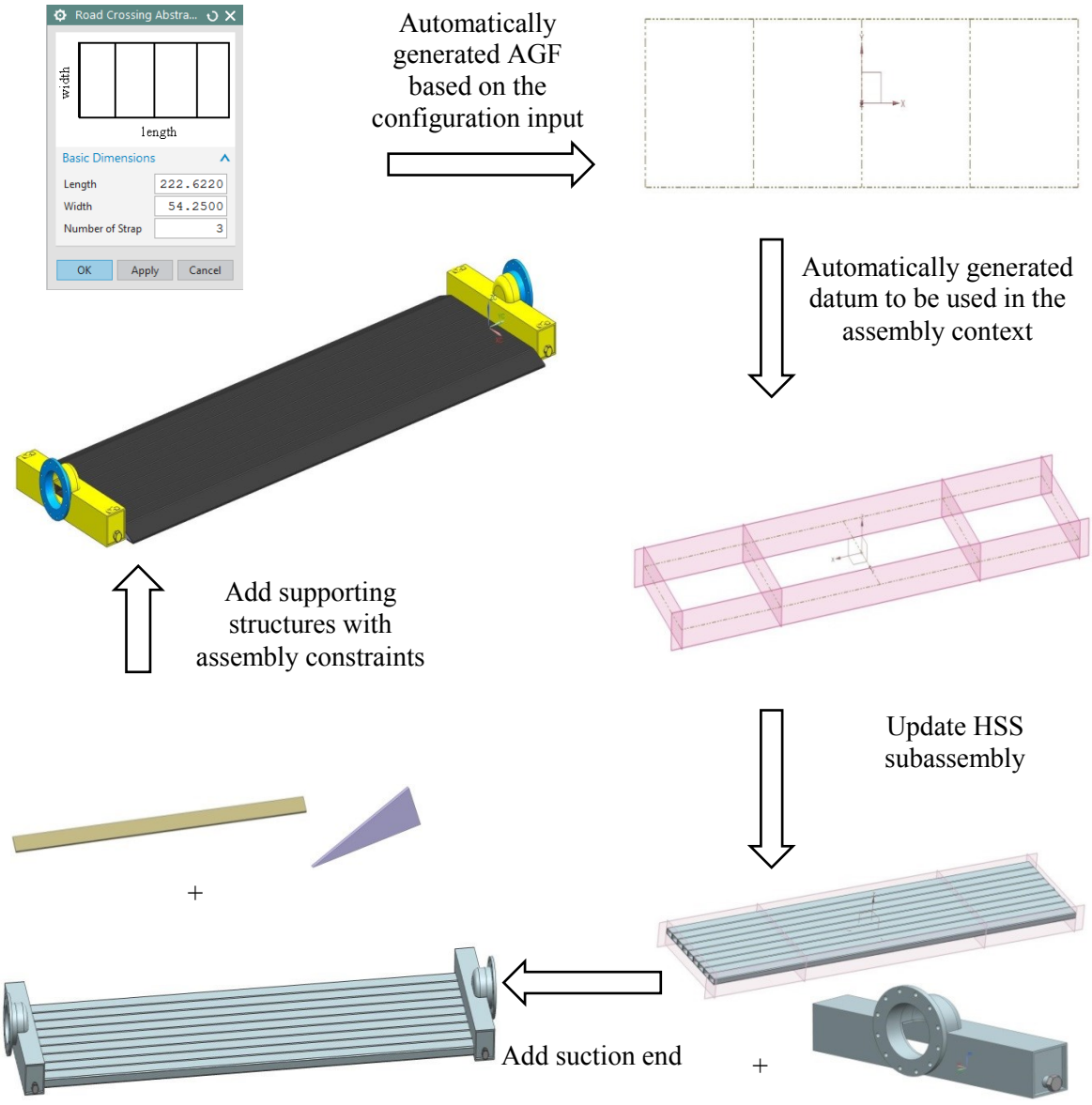


Figure 80 An example workflow with the abstract geometry feature support

8.4 Engine subassembly case study

The engine subassembly has been used extensively in the theoretical exploration of the functional feature modeling. This section takes a step forward and uses it as a practical application of the proposed theory. It is supported by one dedicated GUI to fetch user input based on the predefined application-specific functional feature and two generic GUI to assign

function attribute to faces of interests and visualize the functional faces.

8.4.1 Combustion volume functional feature

In the early stage, parameters based on the application functional feature template can be defined. Figure 81 shows a dedicated application-level GUI serves as guidance and makes it easier for users to type in necessary parameters. The parameters are converted into expressions of the CAD. The conversion is done programmatically. An example piece of code to do so is provided in Figure 82.

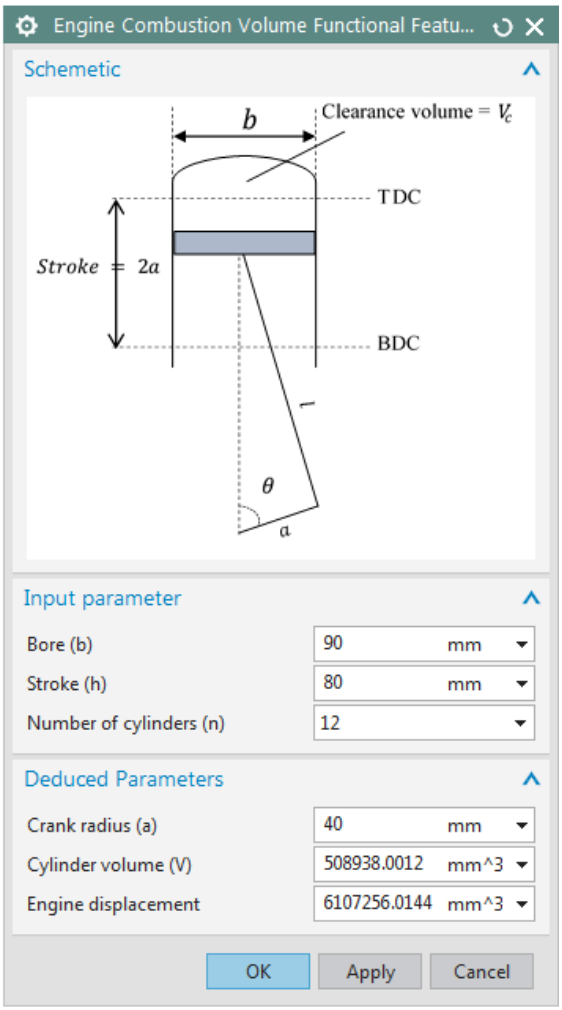


Figure 81 An application-specific GUI for combustion functional feature

```
PropertyList* dia_pl = expression_diameter_FF->GetProperties();
bore = dia_pl->GetDouble("Value");
delete dia_pl;
char tmpBore[32];
sprintf(tmpBore, "%f", dia_pl);
string convertedBore(tmpBore);
NXOpen::Unit *unit1(dynamic_cast<NXOpen::Unit *>(workPart-
>UnitCollection()->FindObject("MilliMeter")));
expression_bore = workPart->Expressions()-
>CreateWithUnits("Bore="+convertedBore, unit1);
```

Figure 82 Example code to create an expression based on the user input

Note that those parameters can be further associated with other aspects of the design, e.g., kinematic performance. Figure 83 gives such an example, relating the functional feature parameters to the position, velocity, and acceleration of the piston.

$$x \approx a\left\{\cos\theta + \frac{l}{a}\left[1 - \frac{1}{2}\left(\frac{a}{l}\right)^2\left(\frac{1}{2} - \frac{1}{2}\cos 2\theta\right)\right]\right\}$$

$$\dot{x} = -a\omega\left(\sin\theta + \frac{a}{2l}\sin 2\theta\right)$$

$$\ddot{x} = -a\omega^2\left(\cos\theta + \frac{a}{l}\cos 2\theta\right)$$

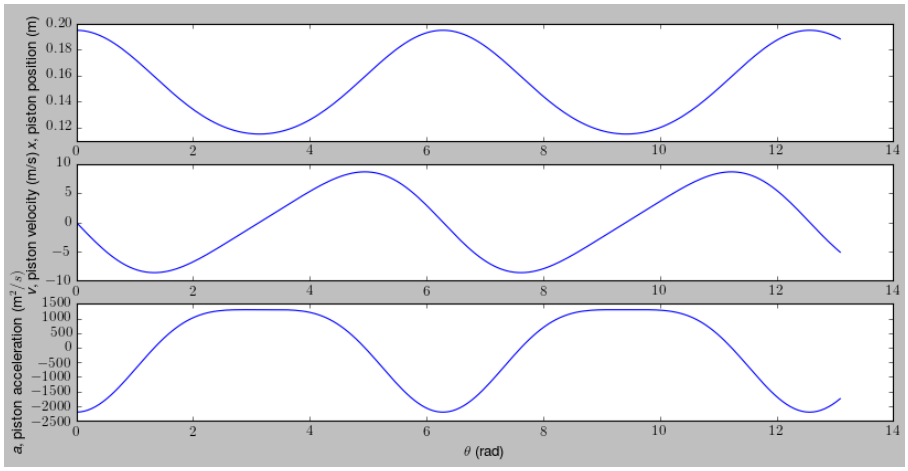


Figure 83 An example of piston position, velocity, and acceleration with respect to θ

The conversion of the functional feature parameters results into a collection of expressions in the system (see Figure 84), which will be used in the downstream design activity, for example, to construct the detailed CAD models. For instance, when constructing the piston part, the

diameter of the top surface is related to the bore defined in the combustion functional feature (Figure 85).

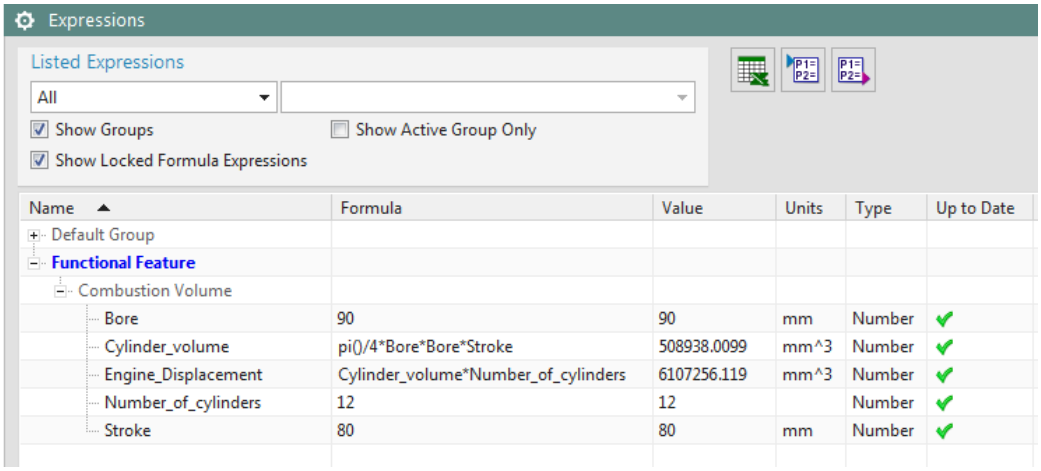


Figure 84 The resulting expressions converted from user inputs

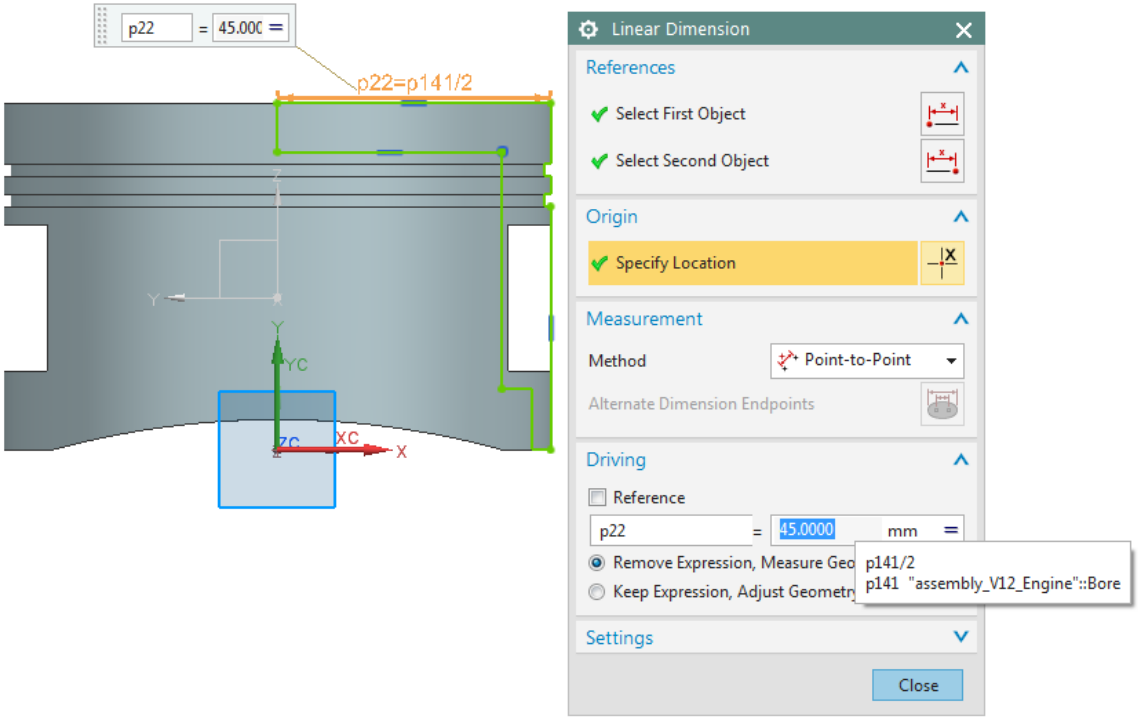


Figure 85 An example shows how to reference a functional feature parameter

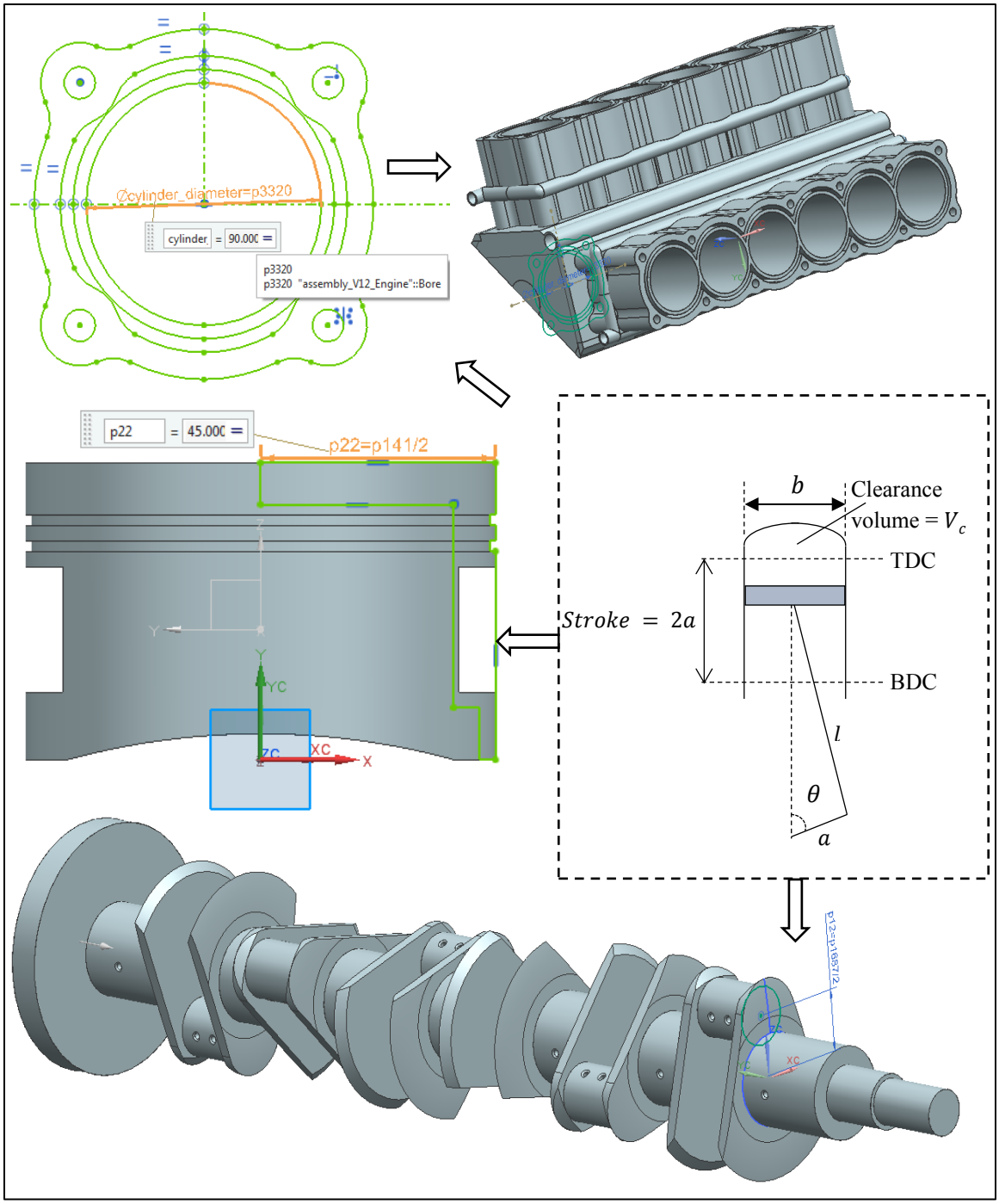


Figure 86 Parts in the engine subassembly that references the functional feature parameters

Similarly, other parts can reference the parameters defined in the combustion functional feature. Figure 86 also shows that the crankcase and crank reference the parameters defined in

the combustion functional feature. The reference relations of the engine assembly to the combustion functional feature are summarized in Figure 87.

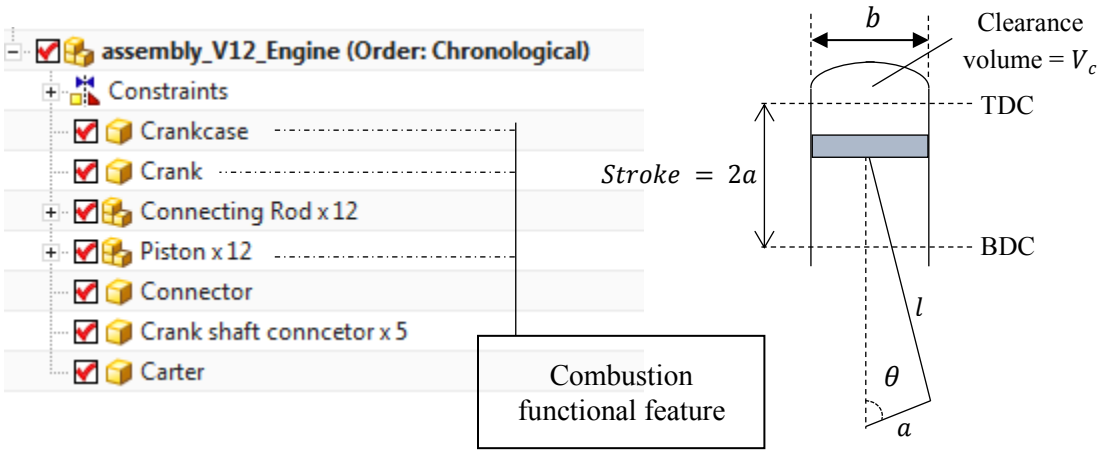


Figure 87 A functional feature relates to an engine subassembly

8.4.2 Capturing functional faces

This subsection presents a prototype for functional faces visualizer. It has been discussed that abstract geometry feature can have different geometric representations and faces are among the key to describe the functional consideration of the design. This subsection shows how to attach function information as attributes to faces of the CAD model, which serves has as a media to convey functional knowledge. Those attributed faces are searched and presented to users to help to convey design intents.

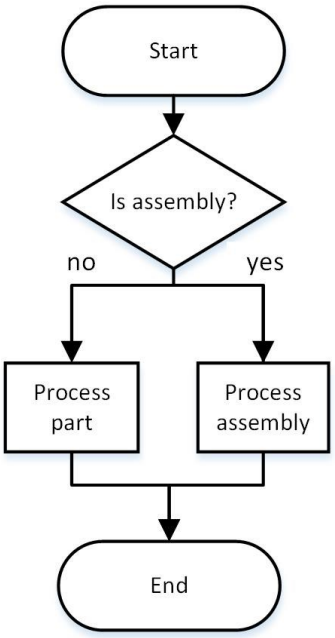
There are three main design considerations for this prototype, which are listed as follows.

- - be able to attach function information to faces of interests
- - be able to recover those faces of interest when necessary
- - need to be generic such that it can work with different models

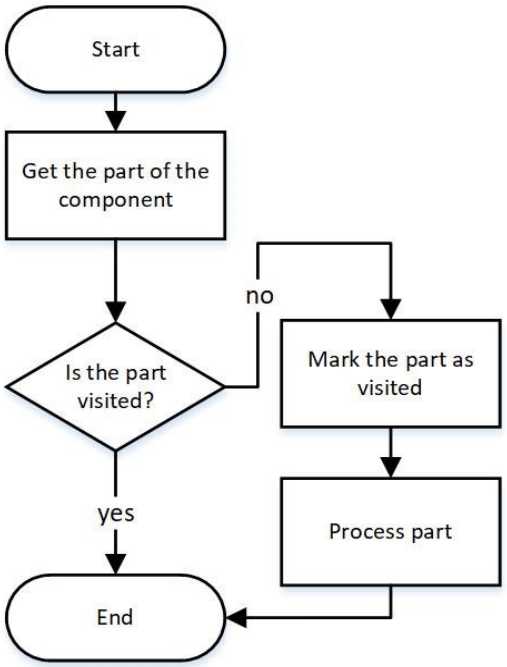
Since CAD modeling is a procedural activity, the prototype should not be an intrusion to end users. After some exploration, it is found that the attribute system in the NX can be to fulfill

the first design consideration. In NX, every *NXObject* can be assigned attributes, which can have different types of values, e.g., number, string, date. Here the objects of interests are some key faces serving different functions, including working faces, auxiliary faces, assembly interfaces. A name-value pair is adopted as the medium to represent a function attribute. It is noted that a group of faces can work collaboratively to fulfill a function. It is also possible that one face is assigned multiple functions.

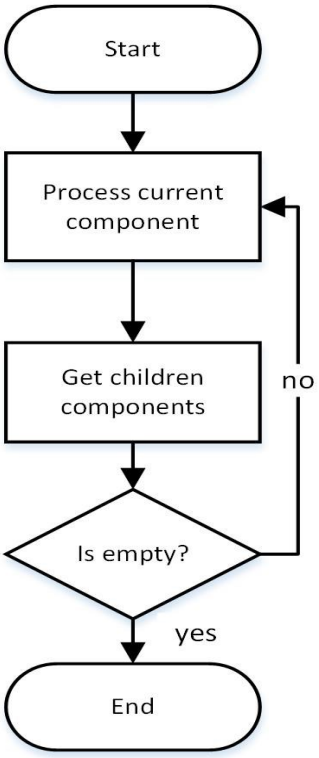
Some algorithmic considerations of searching for the functional faces are described here. Since a CAD model might be only a part or an assembly, the top-level algorithm needs to deals with such cases. If it is a part model, then it can be processed directly. If it is an assembly model, a *recursive* approach is adopted to explore different levels of assembly, i.e., from the top-level assembly to multiple layers of sub-assemblies and to the leaf components. Since we choose to attach function information in the faces of different parts, it needs to find the corresponding part of different components and process the part. When it comes to a part, an *iterative* approach is applied to process all the faces. The flow charts for the algorithms can be found in Figure 88.



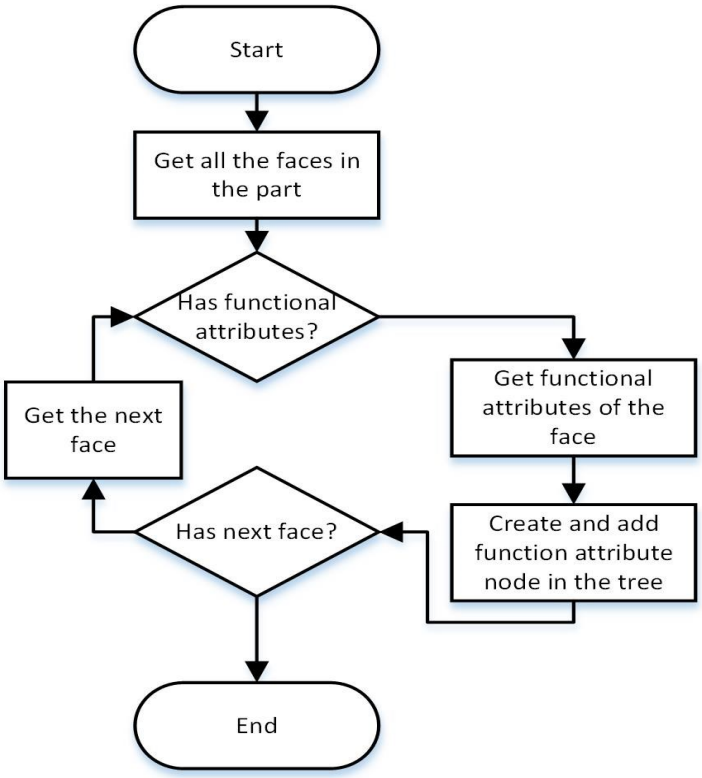
(a) Top level algorithm



(b) Process component



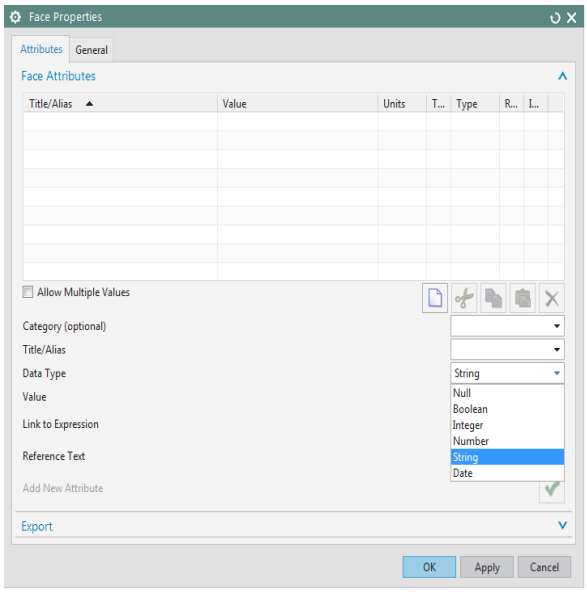
(c) Process assembly



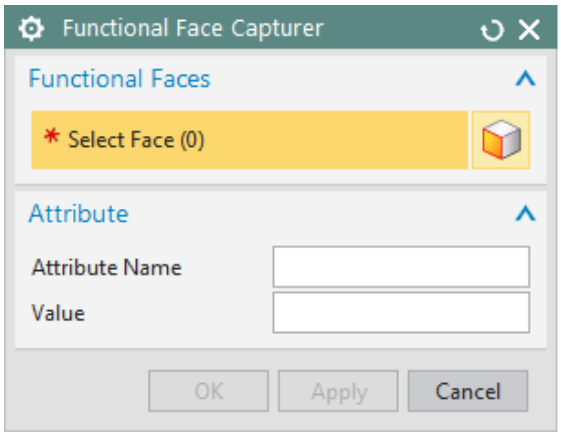
(d) Process part

Figure 88 Key algorithms to search for the attributed faces

There are two approaches to use a UI to assign function information to faces (Figure 89). One is to use the system provided UI to assign function information as attributes to faces of interests. The other is to adopt a dedicated custom UI. The system provided one is more general but might be overkill for the job and it brings some distraction to the end users. The dedicated custom UI is more specific and makes it clear to end users what the functionality is about.



(a) System provided functionality



(b) Custom interface to assign function to a/many faces

Figure 89 Two UI examples to assign function information to faces

The key element is the UI to extract the functional faces, which is presented in Figure 90. Here three tree lists are used as the views toward the functional faces, associated features of the selected face, and the expressions for the feature. Since a feature has parent and children features, here a functionality to populate up and down of the feature is provided. The implementation code for the functional faces visualizer can be found in Appendix 3.

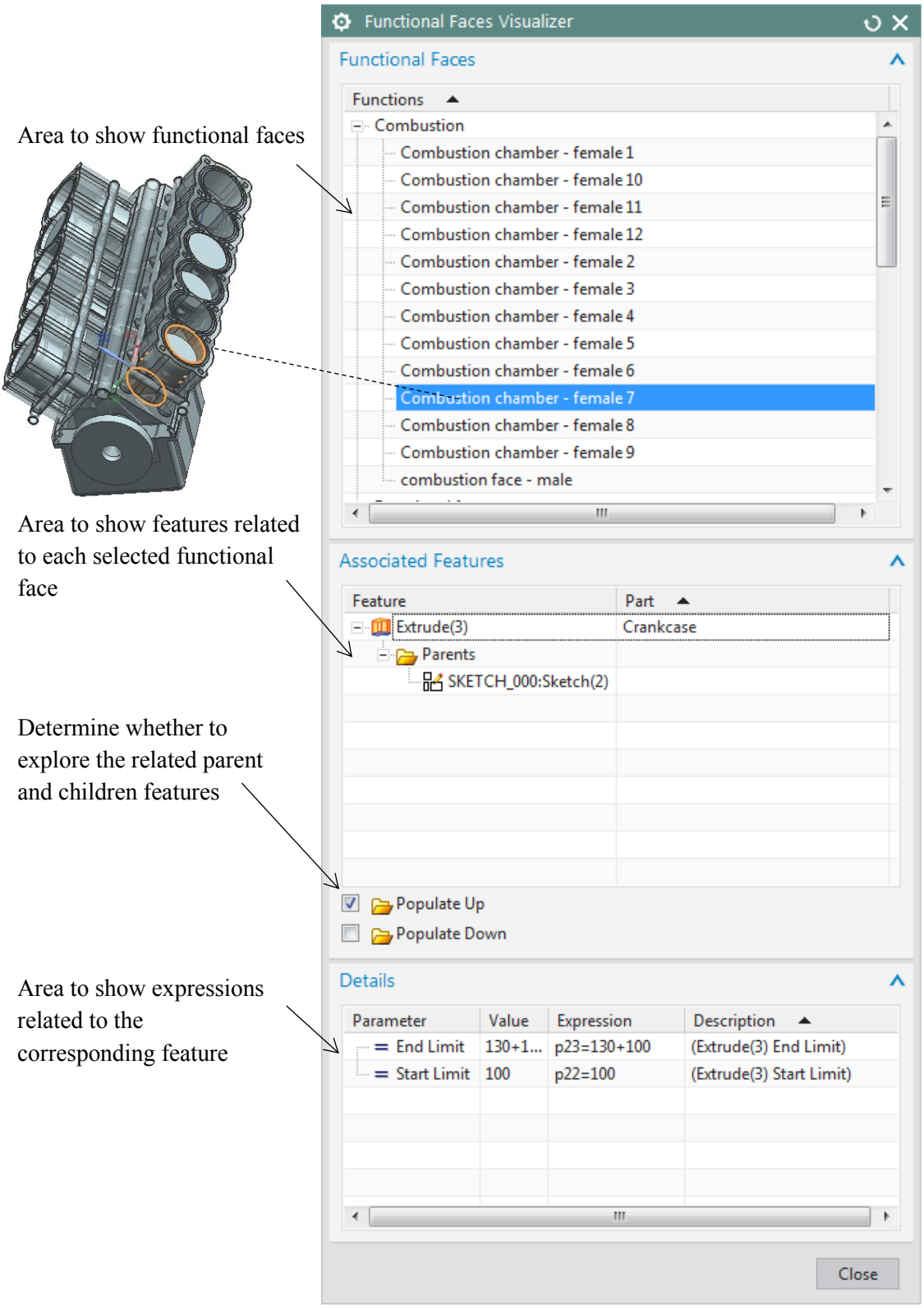


Figure 90 The UI to extract and show the functional faces

Here are the demonstrations for the functional faces visualizer. Figure 91 shows a use case when a group of functional faces is selected in the GUI. In this case, the related faces in the CAD models are highlighted. Note it is likely that those faces belong to different parts of the assembly. Figure 92 presents when a specific functional face is selected, not only the related functional face is highlighted, but also its associated feature is presented with, if possible, its feature expressions. Figure 93 shows how to further explore the parent and children features based on the feature dependency.

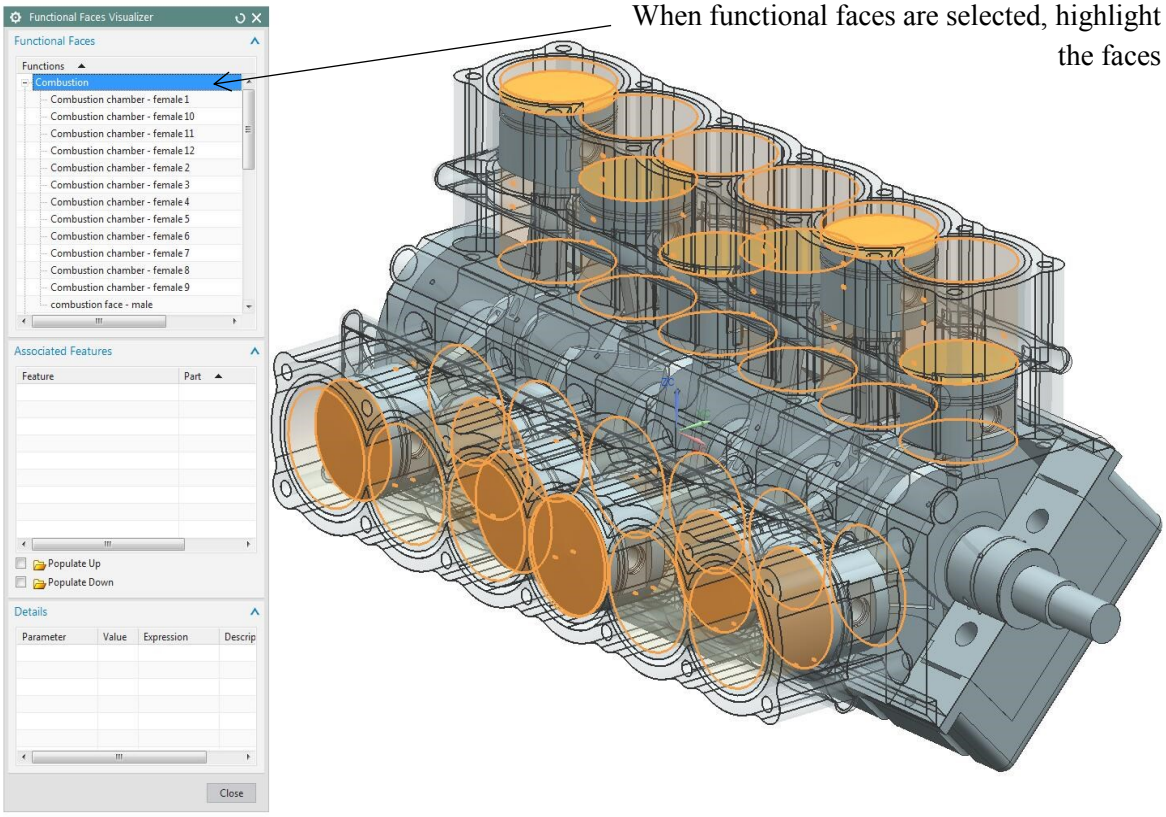


Figure 91 A use case when a group of functional faces is selected

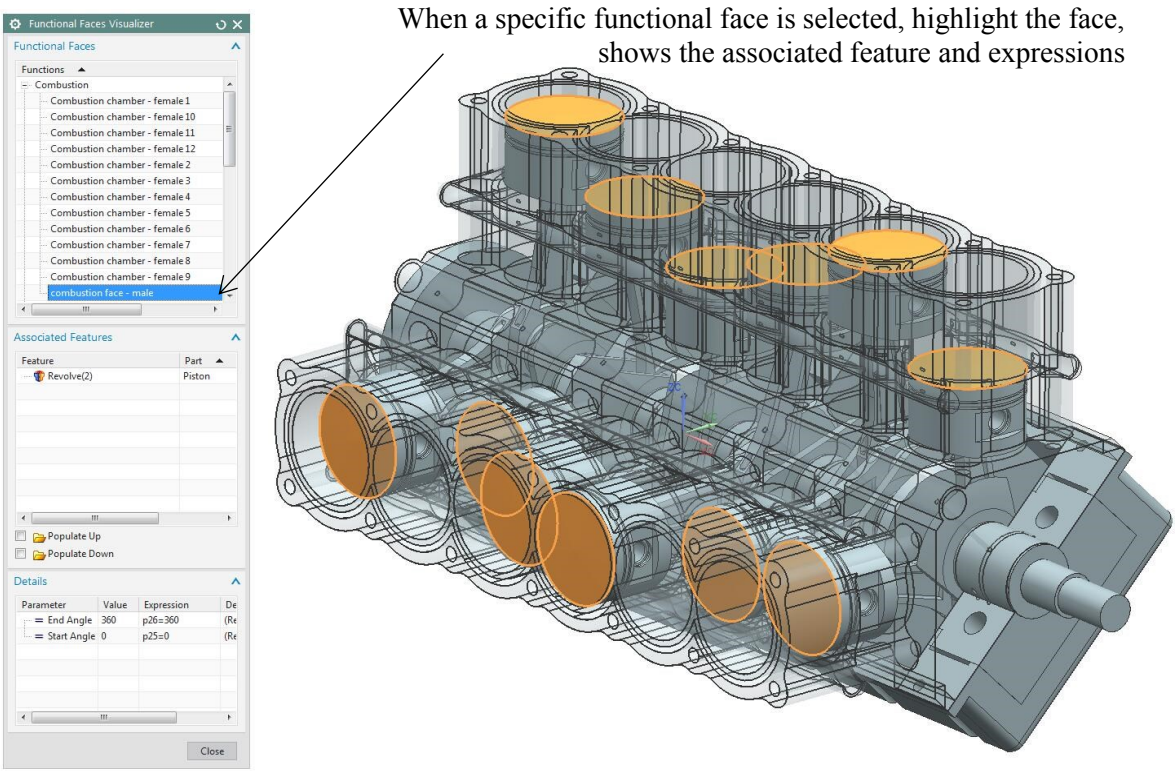


Figure 92 A use case when a specific functional face is selected

8.5 Discussion on the implementation methods

There are two aspects of the implementation method. One is GUI-related design so that users can interact with CAD system, the other is programming to implement the desired functionalities. In terms of the user interface, there are a few options. Siemens NX provides Block UI Styler that users can use to construct the graphical user interfaces. It also generates template code for the GUI, where users can fill in codes for desired functionalities. This approach is applied both in the road crossing and the engine assembly case study.

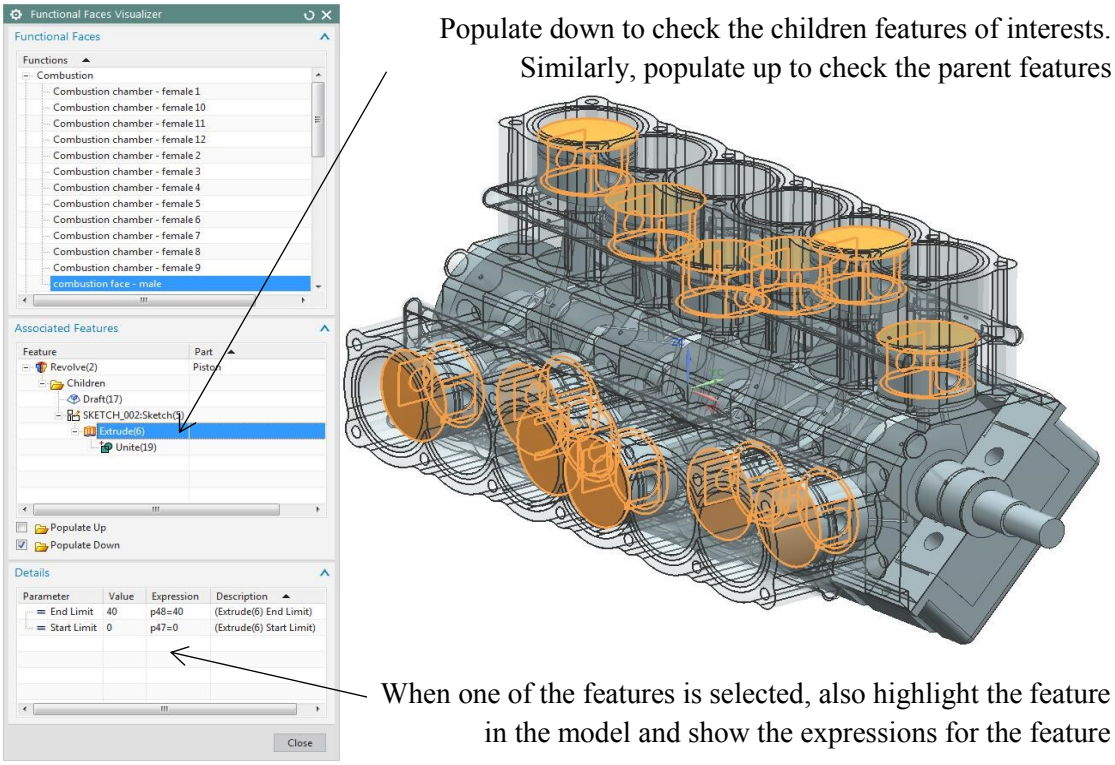


Figure 93 A use case when the associated features are populated

Another approach is to use Product Template Studio, which supports template-based design where users can reuse design information and process knowledge without programming. It is applicable for standardized engineering design. This approach is applied in the slotted liner case study. There are also other approaches, for example, SNAP and Knowledge Fusion, which are applicable depending on the application scenarios and the knowledge of the users. If the readers are interested, they are welcomed to refer to the NX documentation.

What is more important is how to implement the desired functionalities with programming. This requires users to be capable of API programming and have knowledge about the software architecture of NX. For example, which class to use to get the current session, how do feature builders work to construct different features, how to iterate over the features, how to create and manage expressions, where to start and how to travel through the assembly structure, etc.

Fortunately, it can be seen from the API documentation that NX adopts some design pattern, for example, the factory and the builder pattern, which make it easier to work with. In addition, journaling is available for rapid automation. It records NX sessions and the resulting codes can be reused with some modification. With this background knowledge, most functionalities can be achieved with some procedural and object-oriented programming paradigm, together with iterative and recursive algorithms.

8.6 Chapter summary

Functional feature based modeling approach tries to link conceptual functional design with procedural CAD modeling by embedding design knowledge into CAD models, where abstract geometry features, with different geometric representations, are taken as concept carriers with proper constraints and parameterization. It is shown in the slotted liner case study that functional feature is applicable in the part level and in both the road crossing and engine study at the assembly level. Parameterization with expressions plays an important role in functional feature modeling because it is the gateway to a responsive CAD model. These case studies of different levels show that functional feature modeling is achievable.

9 Conclusions and Future Work

9.1 Conclusions

This thesis has gone through a long way from analyzing dependencies among different design elements, which reveals the need of functional design support in CAD. In terms of the functional feature modeling, this thesis focuses on two aspects. Function aspect is the one to start with. Every design has one or more functions to fulfill, be them at product level, module level, or part levels. Functions, other than be categorized into different levels of the design, can also be distinguished by their levels of granularities, i.e., a function might be abstract or concrete. An abstract function can be broken down into a few more concrete ones. Another aspect is the structure. Traditional CAD modeling focuses on the digital prototype of the detailed design structure, which lacks an intermediate layer to facilitate the reasoning of the design. We proposed abstract geometry features as the geometric representation of the functional design considerations, i.e., the functional concept carrier. This section will conclude the thesis with discussions on both the contributions and limitations.

9.1.1 Contributions

From the literature, it is found that functional design is commonly seen in the conceptual design stage of product development and it has limited influence to the CAD modeling domain. The research work on the feature dependency analysis indicates that CAD model construction, although deliver the desired shape of the product, can hardly convey functional design considerations. This thesis argues that it would be helpful to incorporate functional design into CAD model construction such that the resulting models can be knowledge-rich and can convey

design intents. The main theoretical contribution of the proposed methodology of the feature–based functional modeling is that it extends the associative feature with specific ingredients systematically in CAD. This work provides the semantic definitions of the functional features with a concept carrier, i.e., abstract geometry feature, and physics feature.

The detailed and point-by-point contributions are:

- Assessment of the design dependencies in a quantitative and global manner

Information exchanges among different design elements incur design dependencies. The first contribution of this thesis is a network-based approach to assess the design dependency in a quantitative manner such that strategic design decisions could be made to deal with design resources allocation, design activity scheduling, and reduces unnecessary design changes. This approach considers the dependencies of design elements from a global perspective, calculating and ranking the weights of corresponding design elements in terms of their design interaction behaviors. The analysis result of a case from a literature indicates that both functional and structural design considerations exist, which entail more study.

- Assessment of feature dependencies in CAD part models

The feature dependency graph for a part model is created by extracting historical modeling operations and the dependency information of each feature within the part model. It offers a more organized view of the model construction. A posterior analysis of CAD models is proposed to unveil modeling intents by examining feature dependencies with different graph measures, including degree centrality, closeness centrality, betweenness centrality, and eigenvector centrality. It shows critical feature for the construction of the CAD model can be identified with the centrality analysis, which provides engineers a starting point to reexamine the modeling

intents behind the model construction. In the current research, it is found that the reflected modeling intents indicate current CAD modeling practice is geometry-centric and is in need of functional modeling approach.

- A functional concept carrier, abstract geometry, to link functions constructed in the conceptual design stage to the detailed CAD model

This contribution proposes a CAD modeling method based on abstract geometry features to guide designers building CAD models that are valid and agile to represent functional design considerations in the functional feature modeling framework. The modeling of the detailed CAD geometry is based on the synthesis of abstract geometry features, which in turn reflects design functionalities. Functional changes could be traced to abstract geometry features, or the relations among them, and then to the detailed CAD models.

- A generic functional design and modeling approach in CAD environment to associate functional design considerations and the structure of product

Functional design is critical in the engineering design process, but CAD modeling lacks functional design support. Functional feature, and the corresponding semantic definition with the built-in mechanisms, is proposed in this thesis work to support a CAD methodology that models the interrelations between functional and physical considerations of the design and drives the design through the functional perspective. This contribution narrows the gap between functional and geometric representations of the design models.

9.1.2 Research limitations

It is admitted that the proposed functional feature modeling approach is more suitable for product

redesign instead of new product design. The reason behind it is that with product redesign there is adequate knowledge about the product where increment improvement can be made, whereas with new product design it might involve too many iterative efforts.

In the thesis, the functional feature framework contains physics feature and discusses a little bit about it. Admittedly, the exploration is not in-depth. That is an interesting research area where modeling and solving of engineering physics happen. Currently, some CAD systems have integrated CAE capacity to some extent with finite element analysis tools. However, there is still some interoperability issues remained to be solved.

Another limitation of this research is that the proposed methodology, although can be carried out in the existing CAD system with some manual manipulation, lacks a holistic prototype where all these pieces elements could be placed in one place due to the limited access to the CAD system API. There is only so much that can be tailored to a commercial CAD system. For example, in the functional faces visualizer, we have to iterate through all the faces to find the faces with functional attributes, which is time consuming. It would be better if we can store the information in the model directly as an attribute to the part model. This can only happen if we have access to the source code of NX. Nevertheless, the proposed methodology is generic and provides a reasonable starting point for software vendors to implement the details.

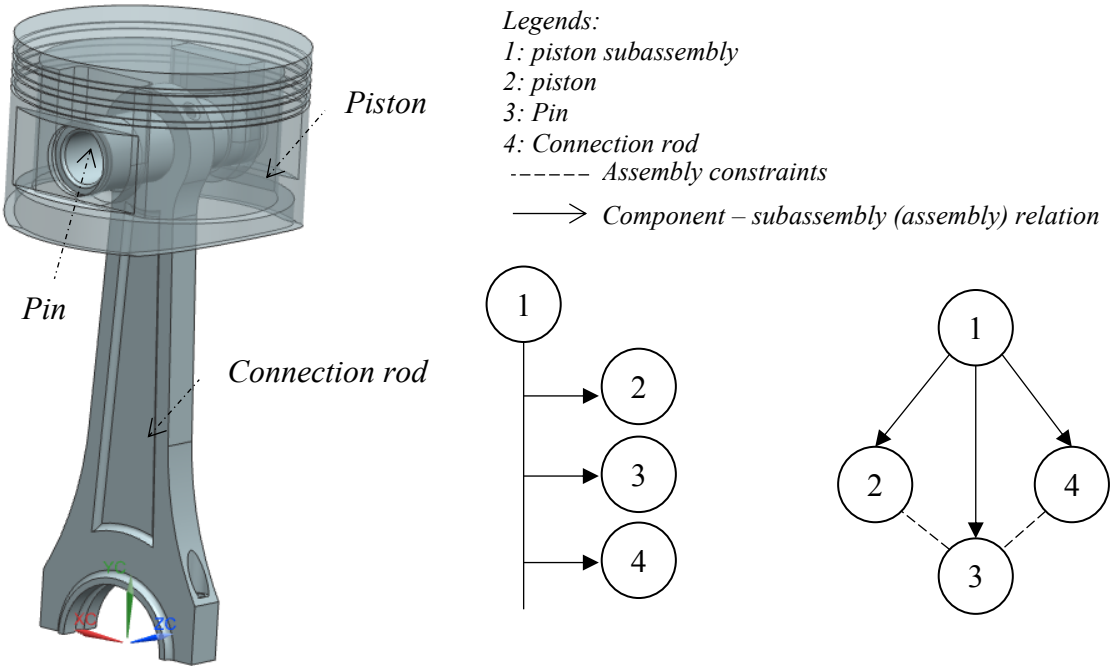
As has been mentioned in the first chapter, different manufacturing processes for a specific design have some impact on the final shape of the design artifacts. That is to say, these parts of the impacts highly depend on different choices of manufacturing processes. The induced functional considerations are mainly manufacturing-oriented. This research does not explore these nuances caused by different manufacturing processes.

9.2 Suggestions for future work

Given the limited time and energy, there are a few directions the author would like to pursue but hasn't done yet. In this section, the author would like to suggest a few possible future work directions that extend the research presented in this thesis.

- Network analysis of product based on assembly graph

Chapter 3 presents a network approach to analyze a product with assembly structure. However, the relations between different components are provided beforehand manually. One possible future direction is to analyze assembly structures by exploring the relations computationally. The relations between components could be assembly constraints. Assembly constraints are used to position different components, including center, concentric, touch, fit, fix, parallel, perpendicular etc. Assembly structure mimics a tree structure with a root component at the top and leaf components on the bottom. Assembly constraints are a kind of assembly relations that could be used to build the assembly graph. For example, in Figure 94, (a) gives a piston subassembly, (b) shows the corresponding tree structure of the piston subassembly, and (c) demonstrates the corresponding assembly graph. Namely, assembly graph not only shows the relations between assembly (subassembly) and components but also indicates assembly constraints between components or subassemblies.



(a) Piston subassembly

(b) Tree structure of the piston subassembly

(c) Assembly graph of the piston subassembly

Figure 94 An example assembly graph with a piston subassembly

- CAD system implementation of functional feature-based modeling procedure

Another possible future research direction is to improve current CAD systems based on the method proposed in chapter 5 and 6 such that they could synthesize the convergence of design functions into solid parts by maintaining the associativity of the abstract geometry features with part modeling operations smoothly. Although the proposed method could be carried out manually by users with existing CAD systems, a systematic implementation method and its detailed guidance to the end users could streamline the cyclic knowledge-rich engineering process and further improve the design efficiency and effectiveness.

In addition, current CAD system has very limited capability for functional design. It can have a module or add-on that helps designers to define functions and carry out functional

decomposition. This is achievable in an object-oriented approach by defining function as a generic class and function structure as a tree-like structure containing different levels of functions. GUI must be provided for users to interact with the module. UI can be constructed using Block UI Styler module in NX. For abstract geometry feature, it is easily doable with NX. WAVE technology can be used to establish the links between abstract geometry feature and the detailed geometry. User interfaces need to be created for the users to aggregate those function and abstract geometry elements into a functional feature

- Integration of functional feature and network analysis

It would be very interesting to integrate the network approach with functional feature modeling. The results of the functional analysis are functions with different granularities. Relations could be built among those functions to see how they are connected. It is clear that hierarchical tree structure is obtainable by breaking down functions layer by layer. What about the relations with functions in the same layer? Once the relations among functions are established for the whole product, a network analysis could be carried out. Possible candidates of interesting network properties include the centralities, density, and propagation paths.

References

Ahmad, Naveed, David C. Wynn, and P. John Clarkson. 2013. “Change Impact on a Product and Its Redesign Process: A Tool for Knowledge Capture and Reuse.” *Research in Engineering Design* 24 (3): 219–44. doi:10.1007/s00163-012-0139-8.

Aifaoui, Nizar, Dominique Deneux, and René Soenen. 2006. “Feature-Based Interoperability between Design and Analysis Processes.” *Journal of Intelligent Manufacturing* 17 (1): 13–27. doi:10.1007/s10845-005-5510-4.

Akman, V, P J W Hagen, and T Tomiyamat. 1990. “A Fundamental and Theoretical Framework for an Intelligent CAD System” 16 (January).

Amadori, Kristian, Mehdi Tarkian, Johan Ölvander, and Petter Krus. 2012. “Flexible and Robust CAD Models for Design Automation.” *Advanced Engineering Informatics* 26 (2). Elsevier Ltd: 180–95. doi:10.1016/j.aei.2012.01.004.

Ando, Sonken, Ryo Ikeda, Hideki Aoyama, and Norihito Hiruma. 2010. “Development of CAD System Based on Function Features.” *Key Engineering Materials* 447–448 (September): 442–46. doi:10.4028/www.scientific.net/KEM.447-448.442.

Ansari, Shadi, Ashker Ibney Rashid, Prashant R Waghmare, Yongsheng Ma, and David Nobes. 2015. “Newtonian and Non - Newtonian Flows through Micro Scale Orifices.” In *10th Pacific Symposium on Flow Visualization and Image Processing*, 15–18. Naples, Italy.

Attene, Marco, Francesco Robbiano, Michela Spagnuolo, and Bianca Falcidieno. 2009. “Characterization of 3D Shape Parts for Semantic Annotation.” *CAD Computer Aided*

Design 41 (10). Elsevier Ltd: 756–63. doi:10.1016/j.cad.2009.01.003.

Berg, Eelco Van Den, Willem F Bronsvoort, and Joris S M Vergeest. 2002. "Freeform Feature Modelling: Concepts and Prospects." *Computer in Industry* 49: 217–33.

Bidarra, R, and W.F F Bronsvoort. 2000. "Semantic Feature Modelling." *Computer-Aided Design* 32 (3): 201–25. doi:10.1016/S0010-4485(99)00090-1.

Blott, Simon J, and Kenneth Pye. 2001. "Gradistat : A Grain Size Distribution and Statistics Package for the Analysis of Unconsolidated Sediments." *Earth Surface Processes and Landforms* 26: 1237–48.

Bluntzer, Jean Bernard, Samuel Gomes, and Jean Claude Sagot. 2008. "From Functional Analysis to Specific Parameters: Description of Knowledge Based CAD Model." *SITIS 2008 - Proceedings of the 4th International Conference on Signal Image Technology and Internet Based Systems*, 665–71. doi:10.1109/SITIS.2008.41.

Bodein, Yannick, Bertrand Rose, and Emmanuel Caillaud. 2014. "Explicit Reference Modeling Methodology in Parametric CAD System." *Computers in Industry* 65 (1). Elsevier B.V.: 136–47. doi:10.1016/j.compind.2013.08.004.

Bonacich, Phillip. 1987. "Power and Centrality: A Family of Measures." *American Journal of Sociology* 92 (5): 1170–82. doi:10.1086/228631.

Bonacich, Phillip, and Paulette Lloyd. 2001. "Eigenvector-like Measures of Centrality for Asymmetric Relations." *Social Networks* 23 (3): 191–201. doi:10.1016/S0378-8733(01)00038-7.

Borgo, Stefano, Massimiliano Carrara, Pawel Garbacz, and Pieter E. Vermaas. 2011. "A Formalization of Functions as Operations on Flows." *Journal of Computing and Information Science in Engineering* 11 (3): 31007. doi:10.1115/1.3615523.

Camba, Jorge, Manuel Contero, Michael Johnson, and Pedro Company. 2014. "Extended 3D Annotations as a New Mechanism to Explicitly Communicate Geometric Design Intent and Increase CAD Model Reusability." *Computer Aided Design* 57: 61–73.

Camba, Jorge D., and Manuel Contero. 2015. "Assessing the Impact of Geometric Design Intent Annotations on Parametric Model Alteration Activities." *Comput. Ind.* 71. Elsevier B.V.: 35–45. doi:10.1016/j.compind.2015.03.006.

Camba, Jorge D, Manuel Contero, and Pedro Company. 2016. "Parametric CAD Modeling: An Analysis of Strategies for Design Reusability." *Computer-Aided Design* 74. Elsevier Ltd: 18–31. doi:10.1016/j.cad.2016.01.003.

Chandrasekaran, B., and John R. Josephson. 2000. "Function in Device Representation." *Eng. Comput.* 16: 162–77.

Chen, G., Y.-S. Ma, G. Thimm, and S.-H. Tang. 2004. "Unified Feature Modeling Scheme for the Integration of CAD and CAx." *Computer-Aided Design and Applications* 1 (1–4): 595–601. doi:10.1080/16864360.2004.10738303.

Chen, G., Y. S. Ma, G. Thimm, and S. H. Tang. 2005. "Knowledge-Based Reasoning in a Unified Feature Modeling Scheme." *Computer-Aided Design and Applications* 2: 173–82.

Cheng, Hui, and Xuening Chu. 2010. "A Network-Based Assessment Approach for Change

Impacts on Complex Product." *Journal of Intelligent Manufacturing* 23 (4): 1419–31. doi:10.1007/s10845-010-0454-8.

Cheng, Zhengrong, and Yongsheng Ma. 2014. "A Network-Based Assessment of Design Dependency in Product Development." In *Proc. 2014 Int. Conf. Innov. Des. Manuf.*, 137–42. Montreal, Quebec, Canada. doi:10.1109/IDAM.2014.6912684.

Deng, Y, G A Britton, and S B Tor. 1998. "A Design Perspective of Mechanical Function and Its Object-Oriented Representation Scheme." *Engineering with Computers* 14: 309–20.

Deng, Yi-min. 2013. "A Behavioural Process Design Model for Development of Assembly Devices Yongsheng Ma" 18 (5): 445–60.

Easley, David, and Jon Kleinberg. 2010. *Networks , Crowds , and Markets : Reasoning about a Highly Connected World*. New York, NY, USA.: Cambridge University Press.

Eckert, Claudia, P. John Clarkson, and Winfried Zanker. 2004. "Change and Customisation in Complex Engineering Domains." *Research in Engineering Design* 15 (1): 1–21. doi:10.1007/s00163-003-0031-7.

Elgh, Fredrik. 2014. "Automated Engineer-to-Order Systems - a Task-Oriented Approach to Enable Traceability of Design Rationale." *International Journal of Agile Systems and Management* 7 (3/4): 324. doi:10.1504/IJASM.2014.065358.

Eppinger, Steven D., and T.R. Browning. 2012. *Design Structure Matrix Methods and Applications*. Cambridge: The MIT Press.

Erden, M.S., H. Komoto, T.J. van Beek, V. D'Amelio, E. Echavarria, and T. Tomiyama. 2008.

"A Review of Function Modeling: Approaches and Applications." *Ai Edam* 22 (2): 147–69. doi:10.1017/S0890060408000103.

Freeman, Linton C. 1977. "A Set of Measures of Centrality Based on Betweenness." *Sociometry* 40 (1): 35–41.

Frillici, Francesco Saverio, Lorenzo Fiorineschi, and Gaetano Cascini. 2015. "Linking TRIZ to Conceptual Design Engineering Approaches." *Procedia Engineering* 0. Elsevier B.V.: 1031–40. doi:10.1016/j.proeng.2015.12.421.

Furui, K., Ding Zhu, Alfred Hill, Eric Davis, and Brian Buck. 2007. "Optimization of Horizontal Well-Completion Design With Cased/Perforated or Slotted Liner Completions." *SPE Production & Operations* 22 (2): 248–53. doi:10.2118/90579-PA.

Furui, Kenji, D. Zhu, and a. Hill. 2005. "A Comprehensive Skin-Factor Model of Horizontal-Well Completion Performance." *SPE Production & Facilities* 20 (3): 207–20. doi:10.2118/84401-PA.

Gao, Shuming, Shuting Zhang, Xiang Chen, and Youdong Yang. 2013. "Computers in Industry A Framework for Collaborative Top-down Assembly Design." *Computers in Industry* 64 (8). Elsevier B.V.: 967–83. doi:10.1016/j.compind.2013.05.007.

Gebhard, Richard. 2013. "A Resilient Modeling Strategy." Solid Edge University.

Gero, John S. 1990. "Design Prototypes : A Knowledge Representation Schema for Design." *AI Magazine* 11 (4): 26–36.

GmbH, Mahel, ed. 2012. *Pistons and Engine Testing*. 1sted. Stuttgart 2012: Vieweg+Teubner

Verlag. doi:10.1007/978-3-8348-8662-0.

Goel, Ashok K., Spencer Rugaber, and Swaroop Vattam. 2009. “Structure, Behavior, and Function of Complex Systems: The Structure, Behavior, and Function Modeling Language.” *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* 23 (1): 23. doi:10.1017/S0890060409000080.

Gruber, TR. 1995. “Toward Principles for the Design of Ontologies Used for Knowledge Sharing.” *International Journal of Human Computer Studies* 43: 907–28.

Habib, Tufail, and Hitoshi Komoto. 2014. “Comparative Analysis of Design Concepts of Mechatronics Systems with a CAD Tool for System Architecting.” *Mechatronics* 24 (0). Elsevier Ltd: 788–804. doi:http://dx.doi.org/10.1016/j.mechatronics.2014.03.003.

Hamraz, Bahram, Nicholas H. M. Caldwell, and P. John Clarkson. 2012. “A Multidomain Engineering Change Propagation Model to Support Uncertainty Reduction and Risk Management in Design.” *Journal of Mechanical Design* 134 (10): 100905. doi:10.1115/1.4007397.

Hamraz, Bahram, Nicholas H M Caldwell, Tom W. Ridgman, and P. John Clarkson. 2015. “FBS Linkage Ontology and Technique to Support Engineering Change Management.” *Research in Engineering Design*, 3–35. doi:10.1007/s00163-014-0181-9.

He, Kunjin, Zhengming Chen, Junfeng Jiang, and Lin Wang. 2014. “Creation of User-Defined Freeform Feature from Surface Models Based on Characteristic Curves.” *Computers in Industry* 65 (4). Elsevier B.V.: 598–609. doi:10.1016/j.compind.2014.01.011.

Hoffmann, Christoph M, and Robert Joan-Arinyo. 1998. “On User-Defined Features.” *Computer-Aided Design* 30 (5): 321–32. doi:10.1016/S0010-4485(97)00048-1.

Hoque, A.S.M, P.K. Halder, M.S. Parvez, and T. Szecsi. 2013. “Integrated Manufacturing Features and Design-for-Manufacture Guidelines for Reducing Product Cost under CAD/CAM Environment.” *Computers and Industrial Engineering* 66 (4). Elsevier Ltd: 988–1003. doi:10.1016/j.cie.2013.08.016.

Huang, G.Q, W.Y Yee, and K.L Mak. 2003. “Current Practice of Engineering Change Management in Hong Kong Manufacturing Industries.” *Journal of Materials Processing Technology* 139 (1–3): 481–87. doi:10.1016/S0924-0136(03)00524-7.

Hubka, Vladimir, and W Ernst Eder. 1988. *Theory of Technical Systems - A Total Concept Theory for Engineering Design*. Berlin, Heidelberg: Springer. doi:https://doi.org/10.1007/978-3-642-52121-8.

Iyer, Ganeshram R., and John J. Mills. 2006. “Design Intent in 2D CAD : Definition and Survey.” *Computer-Aided Design & Application* 3 (1–4): 259–67. doi:10.1080/16864360.2006.10738463.

Jaiswal, Prakhar, Jinmiao Huang, and Rahul Rai. 2014. “Assembly-Based Conceptual 3D Modeling with Unlabeled Components Using Probabilistic Factor Graph.” *CAD Computer Aided Design* 74. Elsevier Ltd: 45–54. doi:10.1016/j.cad.2015.10.002.

Kaiser, T.M.V., S. Wilson, and L.a. Venning. 2002. “Inflow Analysis and Optimization of Slotted Liners.” *SPE Drill. Complet.* 17 (4). doi:10.2118/80145-PA.

Katz, Leo. 1953. “A New Status Index Derived from Sociometric Index.” *Psychometrika* 18 (1): 39–43.

Keuneke, Anne M. 1991. “Device Representation The Significance of Functional Knowledge.” *IEEE Expert* 6 (2): 22–25.

Kitamura, Yoshinobu, Masakazu Kashiwase, Masayoshi Fuse, and Riichiro Mizoguchi. 2004. “Deployment of an Ontological Framework of Functional Design Knowledge.” *Advanced Engineering Informatics* 18 (2): 115–27. doi:10.1016/j.aei.2004.09.002.

Kitamura, Yoshinobu, and Riichiro Mizoguchi. 2003. “Ontology-Based Description of Functional Design Knowledge and Its Use in a Functional Way Server.” *Expert Systems with Applications* 24 (2): 153–66. doi:10.1016/S0957-4174(02)00138-0.

Kleinberg, Jon M. 1999. “Authoritative Sources in a Hyperlinked Environment.” *Journal of the ACM* 46 (5): 604–32. doi:10.1145/324133.324140.

Krumbein, W.C. 1934. “Size Frequency Distributions of Sediments.” *Journal of Sedimentary Petrology* 4: 65–77.

Landers, Diane M., and Pravin Khurana. 2004. Horizontally-structured CAD/CAM modeling for virtual concurrent product and process design. US6775581 B2. *http://www.google.com/patents/EP1243995A3?cl=en*, issued 2004. doi:10.1016/j.(73).

Le, Qize, Zhenghui Sha, and Jitesh H. Panchal. 2014. “A Generative Network Model for Product Evolution.” *Journal of Computing and Information Science in Engineering* 14 (1): 11003. doi:10.1115/1.4025856.

Li, Min, Y. F. Zhang, J. Y. H. Fuh, and Z. M. Qiu. 2009. "Toward Effective Mechanical Design Reuse: CAD Model Retrieval Based on General and Partial Shapes." *Journal of Mechanical Design* 131 (12): 124501. doi:10.1115/1.4000253.

Li, Y G, W Wang, X Liu, and Y S Ma. 2014. "Definition and Recognition of Rib Features in Aircraft Structural Part." *International Journal of Computer Integrated Manufacturing* 27 (1): 1–19. doi:10.1080/0951192X.2013.799784.

Lin, Bor Tsuen, and Shih Hsin Hsu. 2008. "Automated Design System for Drawing Dies." *Expert Systems with Applications* 34 (3): 1586–98. doi:10.1016/j.eswa.2007.01.024.

Lin, Yun. 2008. *Semantic Annotation for Process Models: Facilitating Process Knowledge Management via Semantic Interoperability*. *Diss. Trondheim, Norway*. doi:978-82-471-5160-0.

Liu, Hsu-Chang, and Bartholomew O. Nnaji. 1991. "Design with Spatial Relationships." *Journal of Manufacturing Systems* 10 (6): 449–63. doi:10.1016/0278-6125(91)90003-K.

Ma, Y.-S., G. A. Britton, S. B. Tor, and L. Y. Jin. 2007. "Associative Assembly Design Features: Concept, Implementation and Application." *The International Journal of Advanced Manufacturing Technology* 32 (5–6): 434–44. doi:10.1007/s00170-005-0371-8.

Ma, Y.-S., and Q. Hadi. 2012. "Unified Feature-Based Approach for Process System Design." *International Journal of Computer Integrated Manufacturing* 25 (3): 263–79. doi:10.1080/0951192X.2011.635157.

Ma, Y.-S., and T. Tong. 2003. "Associative Feature Modeling for Concurrent Engineering

Integration." *Computers in Industry* 51 (1): 51–71. doi:10.1016/S0166-3615(03)00025-3.

Ma, Yongsheng. 2013. *Semantic Modeling and Interoperability in Product and Process Engineering*. Edited by Yongsheng Ma. 1sted. Springer Series in Advanced Manufacturing. London: Springer London. doi:10.1007/978-1-4471-5073-2.

MacCormack, Alan, John Rusnak, and Carliss Y. Baldwin. 2006. "Exploring the Structure of Complex Software Designs: An Empirical Study of Open Source and Proprietary Code." *Management Science* 52 (7): 1015–30. doi:10.1287/mnsc.1060.0552.

Mehta, Chandresh, Lalit Patil, and Debasish Dutta. 2013. "An Approach to Determine Important Attributes for Engineering Change Evaluation." *Journal of Mechanical Design* 135 (4): 41003. doi:10.1115/1.4023551.

Miles, LD. 1972. *Techniques of Value Analysis and Engineering*. New York, USA: McGraw-Hill.

Murshed, S M Mahbub, J J Shah, V Jagasivamani, A Wasfy, and D W Hislop. 2007. "OAM+: An Assembly Data Model for Legacy Systems Engineering." In *Proceedings of the ASME International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, DETC2007*, 869–81. doi:10.1115/DETC2007-35723.

Myung, Sehyun, and Soonhung Han. 2001. "Knowledge-Based Parametric Design of Mechanical Products Based on Configuration Design Method." *Expert Systems with Applications* 21: 99–107.

Ouertani, M.Z. 2008. "Supporting Conflict Management in Collaborative Design: An Approach

to Assess Engineering Change Impacts." *Computers in Industry* 59 (9): 882–93. doi:10.1016/j.compind.2008.07.010.

Pahl, Gerhard, Wolfgang Beitz, Jörg Feldhusen, and Karl-Heinrich Grote. 2007. *Engineering Design: A Systematic Approach*. Edited by Ken Wallace and Blessing Lucienne. 3rd ed. London: Springer-Verlag. doi:10.1007/978-1-84628-319-2.

Qureshi, Ahmed Jawad, Jean-Yves Dantan, Jérôme Bruyere, and Régis Bigot. 2010. "Set Based Robust Design of Mechanical Systems Using the Quantifier Constraint Satisfaction Algorithm." *Engineering Applications of Artificial Intelligence* 23: 1173–86. doi:10.1016/j.engappai.2010.02.003.

Rodenacker, W. 1971. *Methodisches Konstruiere*. Heidelberg: Springer-Verlag.

Rouibah, Kamel, and Kevin R. Caskey. 2003. "Change Management in Concurrent Engineering from a Parameter Perspective." *Computers in Industry* 50 (1): 15–34. doi:10.1016/S0166-3615(02)00138-0.

Roy, U., and B. Bharadwaj. 2002. "Design with Part Behaviors: Behavior Model, Representation and Applications." *Computer-Aided Design* 34 (9): 613–36. doi:10.1016/S0010-4485(01)00129-4.

Sanfilippo, Emilio M., and Stefano Borgo. 2016. "What Are Features? An Ontology-Based Review of the Literature." *Computer-Aided Design*. Elsevier Ltd. doi:10.1016/j.cad.2016.07.001.

Schulte, Michael, Christian Weber, and Rainer Stark. 1993. "Functional Features for Design in

Mechanical Engineering." *Computers in Industry* 23 (1–2): 15–24. doi:10.1016/0166-3615(93)90111-D.

Shah, Jami J., and M Mantyla. 1995. *Parametric and Feature-Based CAD/CAM: Concepts, Techniques, and Applications*. New York, USA: John Wiley & Sons.

Shah, Jami J, and Mary T Rogers. 1988. "Functional Requirements and Conceptual Design of the Feature-Based Modeling System." *Computer-Aided Engineering Journal* 5 (1): 9–15.

Suh, N. P. 2001. *Axiomatic Design: Advances and Applications*. New York, USA: Oxford University.

Tao, Songqiao, Zhengdong Huang, Bingquan Zuo, Yangping Peng, and Weirui Kang. 2012. "Partial Retrieval of CAD Models Based on the Gradient Flows in Lie Group." *Pattern Recognition* 45 (4): 1721–38. doi:10.1016/j.patcog.2011.09.017.

Tao, Songqiao, Shuting Wang, and Anhui Chen. 2015. "3D CAD Solid Model Retrieval Based on Region Segmentation." *Multimedia Tools and Applications*. doi:10.1007/s11042-015-3033-3.

Teoh, P C, and K Case. 2004. "Modelling and Reasoning for Failure Modes and Effects Analysis Generation." *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture* 218 (B3): 289–300.

Tomiyama, T., P. Gu, Y. Jin, D. Lutters, Ch Kind, and F. Kimura. 2009. "Design Methodologies: Industrial and Educational Applications." *CIRP Annals - Manufacturing Technology* 58 (2): 543–65. doi:10.1016/j.cirp.2009.09.003.

Tomiyama, Tetsuo. 1994. “From General Design Theory to Knowledge-Intensive Engineering.” *Artificial Intelligence for Engineering, Design, Analysis and Manufacturing* 8 (4): 319–33. doi:10.1017/S0890060400000998.

Torres, V. H., J. Rios, a. Vizan, and J. M. Perez. 2010. “Integration of Design Tools and Knowledge Capture into a CAD System: A Case Study.” *Concurrent Engineering* 18 (4): 311–24. doi:10.1177/1063293X10389788.

Umeda, Y., H. Takeda, T. Tomiyama, and H. Yoshikawa. 1990. “Function, Behaviour, and Structure.” *Applications of Artificial Intelligence in Engineering* 1: 177–93.

Umeda, Yasushi, Masaki Ishii, Masaharu Yoshioka, Yoshiki Shimomura, and Tetsuo Tomiyama. 1996. “Supporting Conceptual Design Based on the Function-Behavior-State Modeler.” *Artificial Intelligence for Engineering, Design, Analysis and Manufacturing* 10 (4): 275–88. doi:10.1017/S0890060400001621.

Umeda, Yasushi, Sinsuke Kondoh, Yoshiki Shimomura, and Tetsuo Tomiyama. 2005. “Development of Design Methodology for Upgradable Products Based on Function–behavior–state Modeling.” *Ai Edam* 19 (3): 161–82. doi:10.1017/S0890060405050122.

Umeda, Yasushi, and Tetsuo Tomiyama. 1997. “Functional Reasoning in Design.” *IEEE Expert* 12 (2): 42–48.

Wan, Renpu. 2011. “Well Completion Mode Selection.” In *Advanced Well Completion Engineering*, Third Edit, 75–116. Elsevier. doi:10.1016/B978-0-12-385868-9.00002-6.

Wright, I.C. 1997. “A Review of Research into Engineering Change Management: Implications

for Product Design." *Design Studies* 18 (1): 33–42. doi:10.1016/S0142-694X(96)00029-4.

Yin, C.-G., and Y.-S. Ma. 2012. "Parametric Feature Constraint Modeling and Mapping in Product Development." *Advanced Engineering Informatics* 26 (3): 539–52. doi:10.1016/j.aei.2012.02.010.

Yuan, Lin, Yusheng Liu, Zhongfei Sun, Yanlong Cao, and Ahsan Qamar. 2016. "A Hybrid Approach for the Automation of Functional Decomposition in Conceptual Design." *Journal of Engineering Design* 4828 (June): 1–28. doi:10.1080/09544828.2016.1146237.

Appendices

Appendix 1

Header file for Feature Finder

```
#ifndef FEATURESFINDER
#define FEATURESFINDER
//FeaturesFinder

// Internal+External Includes
#include <NXOpen/Annotations.hxx>
#include <NXOpen/Assemblies_Component.hxx>
#include <NXOpen/Assemblies_ComponentAssembly.hxx>
#include <NXOpen/Body.hxx>
#include <NXOpen/BodyCollection.hxx>
#include <NXOpen/Face.hxx>
#include <NXOpen/Line.hxx>
#include <NXOpen/NXException.hxx>
#include <NXOpen/NXObject.hxx>
#include <NXOpen/Part.hxx>
#include <NXOpen/PartCollection.hxx>
#include <NXOpen/Session.hxx>
#include <NXOpen/Features_Feature.hxx>
#include <NXOpen/Features_FeatureGroup.hxx>
#include <NXOpen/Features_FeatureCollection.hxx>
#include <NXOpen/Features_BaseFeatureCollection.hxx>
#include <NXOpen/DatumCollection.hxx>
#include <NXOpen/Features_DatumFeature.hxx>
#include <NXOpen/DatumConstraint.hxx>
#include "Helper_openC.h"

//------------------------------------------------------------------------------
---
// NXOpen c++ test class
//------------------------------------------------------------------------------
---
class FeatureFinder
{
    // class members
public:
    static Session *theSession;
    static UI *theUI;

    FeatureFinder();
    ~FeatureFinder();

    void do_it();
    void print(const NXString &);
    void print(const string &);
```

```
    void print(const char*);
    void print(const int);
    TaggedObject* select_by_mask();
    TaggedObject* select_by_type();
    std::vector< NXOpen::TaggedObject * > select_any_objects();
    void getFeatures();
    void getFeatures(int t);

private:
    Part *workPart, *displayPart;
    Selection *selmgr;
    NXMessageBox *mb;
    ListingWindow *lw;
    std::vector<Features::Feature *> features;
    void getDatums();
    static const string sep;

};
#endif
```

CPP file for Feature Finder

```
#include "FeatureFinder.h"
#include "Helper_openC.h"

//------------------------------------------------------------------------------
---
// Initialize static variables
//------------------------------------------------------------------------------
---
Session *(FeatureFinder::theSession) = NULL;
UI *(FeatureFinder::theUI) = NULL;
const string FeatureFinder::sep = ",";
//------------------------------------------------------------------------------
---
// Constructor
//------------------------------------------------------------------------------
---
FeatureFinder::FeatureFinder()
{
    // Initialize the Open C API environment */
//  UF_CALL( UF_initialize() );

    // Initialize the NX Open C++ API environment
    FeatureFinder::theSession = NXOpen::Session::GetSession();
    FeatureFinder::theUI = UI::GetUI();
    selmgr = theUI->SelectionManager();
    mb = theUI->NXMessageBox();
    lw = theSession->ListingWindow();

    workPart = theSession->Parts()->Work();
    displayPart = theSession->Parts()->Display();

}
```

```
//----------------------------------------------------------------------------
---
// Destructor
//----------------------------------------------------------------------------
---
FeatureFinder::~FeatureFinder()
{
//  UF_CALL( UF_terminate() );
}
void FeatureFinder::getFeatures(int t){
//  ListingWindow::DeviceType dtype = ListingWindow::DeviceTypeFileAndWindow;
    string st =  workPart->FullPath().GetText();
    string filename = splitFileName(st);

    filename += "-features";
    filename = newFileName(filename, ".txt");
    std::fstream  fs;
    fs.open(filename, std::fstream::out);

    print(workPart->FullPath());
    Features::FeatureGroup *fgroup;
    workPart->Features()->ConvertToSequentialFeatureGroups();
    fgroup = workPart->Features()->ActiveGroup();
    if(fgroup !=0){
        int fftype = fgroup->GetFeatureGroupType();
        print("feature group type is");
        print(fftype);
    }


    Features::BaseFeatureCollection *fcollections = workPart->BaseFeatures();

    Features::BaseFeatureCollection::iterator it;

    vector<Features::Feature* >::iterator  itt;
    int i = 1;
    print ("new method with feaure collections item : tag :   type    :
name    :    children tag    :    type    :    name ......");
    for(it = fcollections->begin(); it != fcollections->end(); ++it) {
        string item = "";
        stringstream sstag;
        sstag << (*it)->Tag();
        vector<Features::Feature*> childrenfeature = (*it)->GetChildren();
        item += sstag.str() + "," + (*it)->FeatureType().GetText() + ","
+(*it)->GetFeatureName().GetText();;
        for ( itt = childrenfeature.begin(); itt != childrenfeature.end();
++itt) {
            stringstream sctag; // tag for (*itt)
            sctag << (*itt)->Tag();

            item += "," + sctag.str() + "," + (*itt)->FeatureType().GetText()
+"," + (*itt)->GetFeatureName().GetText();
        }
        stringstream iss;
        iss << i;
        item = iss.str() + ","+item;
        i++;
```

```
        fs<< item <<std::endl;
        print(item);
    }
        fs.close();
}
void FeatureFinder::getFeatures(){
//  ListingWindow::DeviceType dtype = ListingWindow::DeviceTypeFileAndWindow;
    string st =  workPart->FullPath().GetText();
    string filename = splitFileName(st);

    filename += "-features";
    filename = newFileName(filename, ".txt");
    std::fstream  fs;
    fs.open(filename, std::fstream::out);

    print(workPart->FullPath());
    features = workPart->Features()->GetFeatures();
    vector<Features::Feature* >::iterator it, itt;
    int i = 1;
    print ("item : tag :   type    :    name    :    children tag    :
type    :    name ......");
    for(it = features.begin(); it != features.end(); ++it) {
        string item = "";
        stringstream sstag;
        sstag << (*it)->Tag();
        vector<Features::Feature*> childrenfeature = (*it)->GetChildren();
        item += sstag.str() + "," + (*it)->FeatureType().GetText() + ","
+(*it)->GetFeatureName().GetText();;
        for ( itt = childrenfeature.begin(); itt != childrenfeature.end();
++itt) {
            stringstream sctag; // tag for (*itt)
            sctag << (*itt)->Tag();

            item += "," + sctag.str() + "," + (*itt)->FeatureType().GetText()
+"," + (*itt)->GetFeatureName().GetText();
        }
        stringstream iss;
        iss << i;
        item = iss.str() + ","+item;
        i++;
        fs<< item <<std::endl;
        print(item);
    }
        fs.close();
}
//----------------------------------------------------------------------------
---
// Print string to listing window or stdout
//----------------------------------------------------------------------------
---
void FeatureFinder::print(const NXString &msg)
{
    if(! lw->IsOpen() ) lw->Open();
    lw->WriteLine(msg);
}
void FeatureFinder::print(const string &msg)
{
```

```
    if(! lw->IsOpen() ) lw->Open();
    lw->WriteLine(msg);
}
void FeatureFinder::print(const int msg)
{
    stringstream ss;
    ss << msg;
    if(! lw->IsOpen() ) lw->Open();
    lw->WriteLine(ss.str());
}


void FeatureFinder::print(const char * msg)
{
    if(! lw->IsOpen() ) lw->Open();
    lw->WriteLine(msg);
}

//----------------------------------------------------------------------------
---
// Selection with mask
//----------------------------------------------------------------------------
---
TaggedObject* FeatureFinder::select_by_mask()
{
    NXString message("Select an object by mask:");
    NXString title("Select object");
    Selection::SelectionScope scope = Selection::SelectionScopeUseDefault;
    Selection::SelectionAction action =
Selection::SelectionActionClearAndEnableSpecific;
    bool include_features = 0;
    bool keep_highlighted = 0;


    std::vector<Selection::MaskTriple> maskArray(1);
    maskArray[0] = Selection::MaskTriple( UF_solid_type,
UF_solid_body_subtype, 0 ); // Bodies
    Point3d cursor;
    TaggedObject *object;

    // Select objects using filter defined by maskArray triples
    Selection::Response res = selmgr->SelectTaggedObject(
            message, title, scope, action, include_features,
            keep_highlighted, maskArray, &object, &cursor );

    if( res == Selection::ResponseObjectSelected )
    {
        return object;
    }
    return 0;
}

//----------------------------------------------------------------------------
---
// Selection with type array
//----------------------------------------------------------------------------
---
```

```
TaggedObject* FeatureFinder::select_by_type()
{
    NXString message("Select an object by type:");
    NXString title("Select object");
    Selection::SelectionScope scope = Selection::SelectionScopeUseDefault;
    bool keep_highlighted = 0;

    std::vector<Selection::SelectionType> typeArray(1);
    typeArray[0] = Selection::SelectionTypeAll;

    Point3d cursor;
    TaggedObject *object;

    // Select objects using filter defined by type array
    Selection::Response res = selmgr->SelectTaggedObject(
            message, title, scope, keep_highlighted,
            typeArray, &object, &cursor );

    if( res == Selection::ResponseObjectSelected )
    {
        return object;
    }
    return 0;
}

//------------------------------------------------------------------------------
---
// Selection any objects
//------------------------------------------------------------------------------
---
std::vector< NXOpen::TaggedObject * > FeatureFinder::select_any_objects()
{
    NXString message("Select any objects:");
    NXString title("Select objects");
    Selection::SelectionScope scope = Selection::SelectionScopeUseDefault;
    bool include_features = 0;
    bool keep_highlighted = 0;
    std::vector< NXOpen::TaggedObject * > objectArray;

    // Select any object array
    Selection::Response res = selmgr->SelectTaggedObjects(
            message, title, scope, include_features,
            keep_highlighted, objectArray );

    return objectArray;
}

void FeatureFinder::getDatums(){
    DatumCollection * datumCollection = workPart->Datums();
    DatumCollection::iterator it ;
    for (it = datumCollection->begin(); it != datumCollection->end(); ++it){
        std::string item = "0\t";
        tag_t item_tag = (*it)->Tag();
        stringstream ss;
        ss << item_tag;
        item += ss.str()+sep+(*it)->Name().GetText();
         (*it)->Highlight();
```

```
        print(item);
    }
}
//-------------------------------------------------------------------------
---
// Do something
//-------------------------------------------------------------------------
---
void FeatureFinder::do_it()
{
    int t=0;
    getFeatures(t);
}
```

Python code to parse the resulting text file and generate graph

```
#!/usr/bin/python
import sys,getopt
import networkx as nx
import matplotlib.pyplot as plt
import pygraphviz as pgv # need pygraphviz or pydot for nx.to_agraph()

def usage():
   print ('feature_finder_graph.py -i <inputfile> -o <outputfile>\n')
   print('example\n', 'python feature_finder_graph.py  -i
input_files/test1.txt -o fig')
def parse_input(argv):
   inputfile = ''
   outputfile = ''
   if not argv:
      print('Please give input file (must) and outputfile(optional, no
extension\n')
      usage()
      sys.exit()
   try:
      opts, args = getopt.getopt(argv,"hi:o:",["ifile=","ofile="])
   except getopt.GetoptError:
      print ('feature_finder_graph.py -i <inputfile> -o <outputfile>')
      sys.exit(2)
   for opt, arg in opts:
      if opt == '-h':
         usage()
         sys.exit()
      elif opt in ("-i", "--ifile"):
         inputfile = arg
      elif opt in ("-o", "--ofile"):
         outputfile = arg
         print ('Input file is ', inputfile)
         print ('Output file is ', outputfile)
   return inputfile, outputfile
def read_input_file(inputfile):  # read from the input file and output
dictionaries
   d = {} #store the graph in a dict where keys are the nodes and values are
the nodes connected with the keys
   dic_tag = {}
   labels = {}
```

```
    f=open(inputfile.strip(),'r')
#    l=f.readline()  # newer version of feature finder no extra line. so no
need to discard it
#    print(l)
    for l in f:
        l= l.strip()
        if l:  # if s is not empty
            s=l.split(',')
            dic_tag[s[1]] = s[0]
            labels[s[0]]= s[0] +': '+ s[2]
            if len(s) >4:
                d[s[1]] = s[4::3]
            else:
                d[s[1]] = []

    d3={}  # insead of using tag, use numbering starting from 1
    for key in d:
        values = d[key]
        if values:
            d3[dic_tag[key]]=[dic_tag[x] for x in values]
        else:
            d3[dic_tag[key]] = []
    f.close()
    return d, dic_tag, d3, labels
def graph_info(g):    # given a graph, printout its information
    print(g.degree())
    g.in_degree()
    g.out_degree()


def output_graph(g,outputfile=None):
    #given a graph, show figure and save

    fig1=plt.figure()
    g.layout(prog='dot')    #['neato'|'dot'|'twopi'|'circo'|'fdp'|'nop']
    if outputfile:
        o = outputfile+'.jpg'
        g.draw(o)

def main(argv):
    inputfile, o = parse_input(argv)
    print(o)
    d1,d2,d3, labels = read_input_file(inputfile)
    #g=nx.DiGraph(d3)
    g=pgv.AGraph(d3,directed=True, dpi=150)
    for node in g.nodes():
        node.attr['label'] = labels[node]
    graph_info(g)
    g.node_attr['fontsize']=15
    output_graph(g, outputfile=o)
    #    print(g.nodes())

    return

if __name__ == "__main__":
    print(sys.argv[1:])
    main(sys.argv[1:])
```

Appendix 2

The full list of centrality values for the connection rod case study.

item	Out degree	Betweenness	Closeness	Eigenvector
1	0.28889	0	0.41927	0.28479
2	0.02222	0	0.2188	0.01622
3	0.11111	0.01995	0.2966	0.04329
4	0.24444	0.01726	0.30566	0.26119
5	0.02222	0	0.07908	0.03356
6	0.04444	0.01709	0.09662	0
7	0.02222	0.00707	0.03333	0
8	0.06667	0.00901	0.07937	0.03768
9	0	0	0	0
10	0	0	0	0
11	0.02222	0.00505	0.02963	0
12	0.02222	0.00202	0.02222	0.01577
13	0.02222	0.00152	0.04444	0.00247
14	0.04444	0.00556	0.05	0
15	0	0	0	0
16	0.02222	0.00202	0.02963	0
17	0.02222	0.00354	0.02222	0
18	0	0	0	0
19	0.02222	0.00269	0.06923	0.01487
20	0.04444	0.01229	0.08366	0
21	0.02222	0.00707	0.03333	0
22	0.06667	0.0032	0.06667	0.04035
23	0.04444	0.0064	0.04444	0.02207
24	0	0	0	0
25	0	0	0	0
26	0.02222	0.00505	0.02963	0
27	0.02222	0.00202	0.02222	0.01577
28	0	0	0	0
29	0.02222	0	0.04537	0
30	0.02222	0.00303	0.04706	0.00247
31	0.02222	0.00505	0.05051	0.00247
32	0.04444	0.00825	0.05926	0.01673
33	0	0	0	0
34	0.02222	0.00067	0.03951	0
35	0.02222	0.00168	0.04	0.01583
36	0.04444	0.00606	0.04444	0.0452
37	0.02222	0	0.075	0.01622
38	0.04444	0.00707	0.09481	0
39	0.02222	0.00303	0.03951	0
40	0.08889	0.00505	0.08889	0.06786
41	0	0	0	0
42	0	0	0	0

43	0.02222	0.00303	0.04	0
44	0.04444	0.00202	0.04444	0.0452
45	0	0	0	0
46	0	0	0	0

Partial map from Index to feature type for the trigger switch (left) and sport car seat model (right)

Index	Feature type
1	DATUM_CSYS
2	DATUM_CSYS
3	SKETCH
4	EXTRUDE
5	DATUM_CSYS
6	SKETCH
7	SWP104
8	TRIM BODY
9	SHELL
10	BLEND
11	DATUM_CSYS
12	SKETCH
13	EXTRUDE
14	EXTRUDE
15	DATUM_CSYS

Index	Feature type
1	DATUM_CSYS
2	DATUM_CSYS
3	SKETCH
4	EXTRUDE
5	BSURF_XFORM
6	MIRROR
7	UNITE
8	BSURF_XFORM
9	BSURF_XFORM
10	BLEND
11	SHELL
12	BLEND
13	BLEND
14	CHAMFER
15	DATUM_PLANE

Appendix 3

Key Code for functional faces visualizer

The main file

```
#include "FunctionalFacesVisualizer.hpp"
//------------------------------------------------------------------------------
extern "C" DllExport void  ufusr(char *param, int *retcod, int param_len)
{
    FunctionalFacesVisualizer *theFunctionalFacesVisualizer = NULL;
    try
    {
        theFunctionalFacesVisualizer = new FunctionalFacesVisualizer();
        // The following method shows the dialog immediately
        theFunctionalFacesVisualizer->Show();
    }
    catch(exception& ex)
    {
        FunctionalFacesVisualizer::theUI->NXMessageBox()->Show("Block
Styler", NXOpen::NXMessageBox::DialogTypeError, ex.what());
    }
    if(theFunctionalFacesVisualizer != NULL)
    {
        delete theFunctionalFacesVisualizer;
        theFunctionalFacesVisualizer = NULL;
    }
}



extern "C" DllExport int ufusr_ask_unload()
{
    //return (int)Session::LibraryUnloadOptionExplicitly;
    return (int)Session::LibraryUnloadOptionImmediately;
    //return (int)Session::LibraryUnloadOptionAtTermination;
}

//------------------------------------------------------------------------------
// Following method cleanup any housekeeping chores that may be needed.
// This method is automatically called by NX.
//------------------------------------------------------------------------------
extern "C" DllExport void ufusr_cleanup(void)
{
    try
    {
    }
    catch(exception& ex)
    {
        FunctionalFacesVisualizer::theUI->NXMessageBox()->Show("Block
Styler", NXOpen::NXMessageBox::DialogTypeError, ex.what());
    }
}
```

Header file for Functional Faces Visualizer

```
#ifndef FUNCTIONALFACESVISUALIZER_H_INCLUDED
#define FUNCTIONALFACESVISUALIZER_H_INCLUDED

//------------------------------------------------------------------------------
---
//These includes are needed for the following template code
//------------------------------------------------------------------------------
---
#include <uf_defs.h>
#include <uf_ui_types.h>
#include <iostream>
#include <NXOpen/Session.hxx>
#include <NXOpen/UI.hxx>
#include <NXOpen/NXMessageBox.hxx>
#include <NXOpen/Callback.hxx>
#include <NXOpen/NXException.hxx>
#include <NXOpen/BlockStyler_UIBlock.hxx>
#include <NXOpen/BlockStyler_BlockDialog.hxx>
#include <NXOpen/BlockStyler_PropertyList.hxx>
#include <NXOpen/BlockStyler_Node.hxx>
#include <NXOpen/BlockStyler_Group.hxx>
#include <NXOpen/BlockStyler_Tree.hxx>
#include <NXOpen/BlockStyler_Toggle.hxx>
#include <NXOpen/NXObject.hxx>
#include <NXOpen/Part.hxx>
#include <NXOpen/PartCollection.hxx>
#include <NXOpen/Assemblies_Component.hxx>
#include <NXOpen/Assemblies_ComponentAssembly.hxx>
#include <NXOpen/Body.hxx>
#include <NXOpen/BodyCollection.hxx>
#include <NXOpen/Face.hxx>
#include <map>
#include <set>
#include "utilities.hpp"
//------------------------------------------------------------------------------
---
// Namespaces needed for following template
//------------------------------------------------------------------------------
---
using namespace std;
using namespace NXOpen;
using namespace NXOpen::BlockStyler;

class DllExport FunctionalFacesVisualizer
{
    // class members
public:
    static Session *theSession;
    static UI *theUI;
    FunctionalFacesVisualizer();
    ~FunctionalFacesVisualizer();
    int Show();

    void initialize_cb();
```

```
    void dialogShown_cb();
    int update_cb(NXOpen::BlockStyler::UIBlock* block);
    PropertyList* GetBlockProperties(const char *blockID);


    void    OnSelectCallback_AttributeTree(NXOpen::BlockStyler::Tree    *tree,
NXOpen::BlockStyler::Node *, int columnID, bool selected);
    void     OnSelectCallback_FeatureTree(NXOpen::BlockStyler::Tree     *tree,
NXOpen::BlockStyler::Node *, int columnID, bool selected);
    NXOpen::BlockStyler::Tree::ControlType
AskEditControlCallback_ExpressionTree(NXOpen::BlockStyler::Tree         *tree,
NXOpen::BlockStyler::Node *node, int columnID);
    NXOpen::BlockStyler::Tree::BeginLabelEditState
OnBeginLabelEditCallback_ExpressionTree(NXOpen::BlockStyler::Tree       *tree,
NXOpen::BlockStyler::Node *node, int columnID);
    NXOpen::BlockStyler::Tree::EndLabelEditState
OnEndLabelEditCallback_ExpressionTree(NXOpen::BlockStyler::Tree         *tree,
NXOpen::BlockStyler::Node *node, int, NXString editedText);
    void     OnMenuCallback_AttributeTree(NXOpen::BlockStyler::Tree     *tree,
NXOpen::BlockStyler::Node *node, int menuItemID);
    void       OnMenuSelectionCallback_AttributeTree(NXOpen::BlockStyler::Tree
*tree, NXOpen::BlockStyler::Node *node, int menuItemID);


private:
    const char* theDlxFileName;
    NXOpen::BlockStyler::BlockDialog* theDialog;
    NXOpen::BlockStyler::Group* functionAttributeGroup;// Block type: Group
    NXOpen::BlockStyler::Tree*  attributes_tree_control0;//  Block  type:  Tree
Control
    NXOpen::BlockStyler::Group* featureGroup;// Block type: Group
    NXOpen::BlockStyler::Tree*  feature_tree_control01;//  Block  type:  Tree
Control
    NXOpen::BlockStyler::Toggle* toggle_populate_up;// Block type: Toggle
    NXOpen::BlockStyler::Toggle* toggle_populate_down;// Block type: Toggle
    NXOpen::BlockStyler::Group* detailedExpressionsgroup;// Block type: Group
    NXOpen::BlockStyler::Tree* expressions_tree_control02;// Block type: Tree
Control
    BlockStyler::Node*   CreateAndAddFunctionAttributeNode(const   std::string
&name,   const   std::string   &   value,   const   NXObject::AttributeType&   ,
BlockStyler::Node* afterNode, Face* face, Expression* );
    BlockStyler::Node*        CreateAssociatedFeatureNodes(Face*        face,
BlockStyler::Node* afterNode);
    BlockStyler::Node*    CreateAssociatedExpressionNodes(/*BlockStyler::Node*
featureNode,*/ Features::Feature*, BlockStyler::Node* afterNode);
    BlockStyler::Node*       EditAssociatedExpressionNodes(BlockStyler::Node*
featureNode, Expression* exp);
    //void redraw();
    void     PopulateFeatureTreeUp(Features::Feature*,     BlockStyler::Node*
parentNode);
    void    PopulateFeatureTreeDown(Features::Feature*,    BlockStyler::Node*
parentNode);

    std::map<std::string, tag_t> nodes_map;
    void SearchFunctionalFaces();
```

```
    void SearchFunctionalFaces_helper(Assemblies::Component*, std::set<tag_t>
&);
    void        SearchFunctionalFacesInComponent_helper(Assemblies::Component*
comp, std::set<tag_t> &visited_parts);

    void ProcessFace(Part* part, Face*);
    Part* workpart;
    bool isAssembly;
    bool isPopulateUp, isPopulateDown;
    //set<tag_t> nodes_with_expression;
    multimap<tag_t, tag_t> expressionToNodeMap;
    void DeleteUserAttribute_AttributeTreeNode(BlockStyler::Node*);
};
#endif //FUNCTIONALFACESVISUALIZER_H_INCLUDED
```

CPP file for Functional Faces Visualizer

```
//------------------------------------------------------------------------
---
//These includes are needed for the following template code
//------------------------------------------------------------------------
---
#include "FunctionalFacesVisualizer.hpp"
#include <vector>
#include <NXOpen/Face.hxx>
#include <NXOpen/Features_FeatureCollection.hxx>
#include <NXOpen/Features_Feature.hxx>
#include <NXOpen/BasePart.hxx>
#include <NXOpen/Part.hxx>
#include <NXOpen/PartCollection.hxx>
#include <set>
#include <sstream>
#include <NXOpen/Expression.hxx>
#include <NXOpen/ExpressionCollection.hxx>
#include <NXOpen/NXObjectManager.hxx>
#include <NXOpen/LoadOptions.hxx>
#include <NXOpen/AttributePropertiesBuilder.hxx>
#include <NXOpen/AttributeManager.hxx>
#include <NXOpen/ObjectGeneralPropertiesBuilder.hxx>
#include <NXOpen/PropertiesManager.hxx>

using namespace NXOpen;
using namespace NXOpen::BlockStyler;


//------------------------------------------------------------------------
---
// Initialize static variables
//------------------------------------------------------------------------
---
Session *(FunctionalFacesVisualizer::theSession) = NULL;
UI *(FunctionalFacesVisualizer::theUI) = NULL;
//------------------------------------------------------------------------
---
// Constructor for NX Styler class
```

```
//------------------------------------------------------------------------------
---
FunctionalFacesVisualizer::FunctionalFacesVisualizer()
{
    try
    {
        // Initialize the NX Open C++ API environment
        FunctionalFacesVisualizer::theSession                                    =
NXOpen::Session::GetSession();
        FunctionalFacesVisualizer::theUI = UI::GetUI();
        theDlxFileName = "FunctionalFacesVisualizer.dlx";
        theDialog              =              FunctionalFacesVisualizer::theUI-
>CreateDialog(theDlxFileName);
        // Registration of callback functions
        theDialog->AddUpdateHandler(make_callback(this,
&FunctionalFacesVisualizer::update_cb));
        theDialog->AddInitializeHandler(make_callback(this,
&FunctionalFacesVisualizer::initialize_cb));
        theDialog->AddDialogShownHandler(make_callback(this,
&FunctionalFacesVisualizer::dialogShown_cb));
        workpart = theSession->Parts()->Work();
        isAssembly = false;
        theSession->Parts()->LoadOptions()->SetUsePartialLoading(false);

    }
    catch(exception& ex)
    {
        throw;
    }
}

//------------------------------------------------------------------------------
---
// Destructor for NX Styler class
//------------------------------------------------------------------------------
---
FunctionalFacesVisualizer::~FunctionalFacesVisualizer()
{
    if (theDialog != NULL)
    {
        delete theDialog;
        theDialog = NULL;
    }
}


int FunctionalFacesVisualizer::Show()
{
    try
    {
        theDialog->Show();
    }
    catch(exception& ex)
    {
        FunctionalFacesVisualizer::theUI->NXMessageBox()->Show("Block
Styler", NXOpen::NXMessageBox::DialogTypeError, ex.what());
    }
```

```
    return 0;
}

//------------------------------------------------------------------------------
//----------------------Block UI Styler Callback Functions--------------------
//------------------------------------------------------------------------------

//------------------------------------------------------------------------------
//Callback Name: initialize_cb
//------------------------------------------------------------------------------
void FunctionalFacesVisualizer::initialize_cb()
{
    try
    {
        functionAttributeGroup                                                  =
dynamic_cast<NXOpen::BlockStyler::Group*>(theDialog->TopBlock()-
>FindBlock("functionAttributeGroup"));
        attributes_tree_control0                                                =
dynamic_cast<NXOpen::BlockStyler::Tree*>(theDialog->TopBlock()-
>FindBlock("attributes_tree_control0"));
        featureGroup   =   dynamic_cast<NXOpen::BlockStyler::Group*>(theDialog-
>TopBlock()->FindBlock("featureGroup"));
        feature_tree_control01                                                  =
dynamic_cast<NXOpen::BlockStyler::Tree*>(theDialog->TopBlock()-
>FindBlock("feature_tree_control01"));
        toggle_populate_up                                                      =
dynamic_cast<NXOpen::BlockStyler::Toggle*>(theDialog->TopBlock()-
>FindBlock("toggle_populate_up"));
        toggle_populate_down                                                    =
dynamic_cast<NXOpen::BlockStyler::Toggle*>(theDialog->TopBlock()-
>FindBlock("toggle_populate_down"));
        detailedExpressionsgroup                                                =
dynamic_cast<NXOpen::BlockStyler::Group*>(theDialog->TopBlock()-
>FindBlock("detailedExpressionsgroup"));
        expressions_tree_control02                                              =
dynamic_cast<NXOpen::BlockStyler::Tree*>(theDialog->TopBlock()-
>FindBlock("expressions_tree_control02"));

        attributes_tree_control0->SetOnSelectHandler(make_callback(this,
&FunctionalFacesVisualizer::OnSelectCallback_AttributeTree));

        attributes_tree_control0->SetOnMenuHandler(make_callback(this,
&FunctionalFacesVisualizer::OnMenuCallback_AttributeTree));

        attributes_tree_control0-
>SetOnMenuSelectionHandler(make_callback(this,
&FunctionalFacesVisualizer::OnMenuSelectionCallback_AttributeTree));


        feature_tree_control01->SetOnSelectHandler(make_callback(this,
&FunctionalFacesVisualizer::OnSelectCallback_FeatureTree));


         expressions_tree_control02-
>SetOnBeginLabelEditHandler(make_callback(this,
&FunctionalFacesVisualizer::OnBeginLabelEditCallback_ExpressionTree));
```

```
        expressions_tree_control02-
>SetOnEndLabelEditHandler(make_callback(this,
&FunctionalFacesVisualizer::OnEndLabelEditCallback_ExpressionTree));

    }
    catch(exception& ex)
    {
        FunctionalFacesVisualizer::theUI->NXMessageBox()->Show("Block
Styler", NXOpen::NXMessageBox::DialogTypeError, ex.what());
    }
}

enum ExpressionColumns{
    ExpressionColumn1 = 0,
    ExpressionColumn2 = 1,
    ExpressionColumn3 = 2,
    ExpressionColumn4 = 3,

};

enum FeatureColumns{
    FeatureColumn1 = 0
};
//----------------------------------------------------------------------------
---
//Callback Name: dialogShown_cb
//This callback is executed just before the dialog launch. Thus any value set
//here will take precedence and dialog will be launched showing that value.
//----------------------------------------------------------------------------
---
void FunctionalFacesVisualizer::dialogShown_cb()
{
    try
    {
        attributes_tree_control0->InsertColumn(0, "Functions", 400);

        feature_tree_control01->InsertColumn(0, "Feature", 200);
        feature_tree_control01->InsertColumn(1, "Part", 200);

        feature_tree_control01->SetShowHeader(true);
        expressions_tree_control02->InsertColumn(ExpressionColumn1,
"Parameter", 100);
        expressions_tree_control02->InsertColumn(ExpressionColumn2,   "Value",
50);
        expressions_tree_control02->InsertColumn(ExpressionColumn3,
"Expression", 100);
        expressions_tree_control02->InsertColumn(ExpressionColumn4,
"Description", 100);

                //Set resize policy for columns
        expressions_tree_control02->SetColumnResizePolicy(ExpressionColumn1,
Tree::ColumnResizePolicyConstantWidth);
        expressions_tree_control02->SetColumnResizePolicy(ExpressionColumn2,
Tree::ColumnResizePolicyConstantWidth);
        expressions_tree_control02->SetColumnResizePolicy(ExpressionColumn3,
Tree::ColumnResizePolicyConstantWidth);
```

```
        expressions_tree_control02->SetColumnResizePolicy(ExpressionColumn4,
Tree::ColumnResizePolicyConstantWidth);

        attributes_tree_control0->SetHeight(200);
        feature_tree_control01->SetHeight(200);
        expressions_tree_control02->SetHeight(150);
        auto togglePropsDown = toggle_populate_down->GetProperties();
        isPopulateDown = togglePropsDown->GetLogical("Value");
        delete togglePropsDown;
        auto togglePropsUp = toggle_populate_up->GetProperties();
        isPopulateUp = togglePropsUp->GetLogical("Value");
        delete togglePropsUp;

        SearchFunctionalFaces();
    }
    catch(exception& ex)
    {
        FunctionalFacesVisualizer::theUI->NXMessageBox()->Show("Block
Styler", NXOpen::NXMessageBox::DialogTypeError, ex.what());
    }
}

//------------------------------------------------------------------------------
---
//Callback Name: update_cb
//------------------------------------------------------------------------------
---
int FunctionalFacesVisualizer::update_cb(NXOpen::BlockStyler::UIBlock* block)
{
    try
    {
        if (block == toggle_populate_down) {
            auto togglePropsProps = toggle_populate_down->GetProperties();
            isPopulateDown = togglePropsProps->GetLogical("Value");
            delete togglePropsProps;
        }

        else if (block == toggle_populate_up) {
            auto togglePropsProps = toggle_populate_up->GetProperties();
            isPopulateUp = togglePropsProps->GetLogical("Value");
            delete togglePropsProps;
        }
    }
    catch(exception& ex)
    {
        FunctionalFacesVisualizer::theUI->NXMessageBox()->Show("Block
Styler", NXOpen::NXMessageBox::DialogTypeError, ex.what());
    }
    return 0;
}

//------------------------------------------------------------------------------
---
//Function Name: GetBlockProperties
//Description: Returns the propertylist of the specified BlockID
//------------------------------------------------------------------------------
---
```

```
PropertyList*     FunctionalFacesVisualizer::GetBlockProperties(const     char
*blockID)
{
    return theDialog->GetBlockProperties(blockID);
}

//---------------------------------------------------------------------------
---
//Function Name: CreateAndAddFunctionAttributeNode
//Description: Create and add node to the function attribute tree and put the
associated face* to the data container
//---------------------------------------------------------------------------
---

BlockStyler::Node*
FunctionalFacesVisualizer::CreateAndAddFunctionAttributeNode(const
std::string &name, const std::string & value,const NXObject::AttributeType&
attributeType, BlockStyler::Node* afterNode, Face* face, Expression* exp)
{
    BlockStyler::Node* parentNode;
    //string nodeName;
    auto it = nodes_map.find(name);
    // Need to add face to the parent ndoe?
    if (it == nodes_map.end()) { // need to create a new parent node and a
child node
        parentNode = NULL;
        BlockStyler::Node*       node       =       attributes_tree_control0-
>CreateNode(name.c_str());
        DataContainer* parentDatacontainer = node->GetNodeData();
        std::vector<TaggedObject*> faces;
        faces.push_back(face);
        parentDatacontainer->AddTaggedObjectVector("Data", faces); //property
name is Data for the primary name. this primary Data is used by NX for some
operation such as cross selection
        // update nodes_map
        nodes_map[name] = node->Tag();
        attributes_tree_control0->InsertNode(node,   parentNode,   afterNode,
attributes_tree_control0->NodeInsertOptionSort);
        //node->ScrollTo(0,BlockStyler::Node::ScrollCenter);


        BlockStyler::Node*       cnode       =       attributes_tree_control0-
>CreateNode(value.c_str());
        DataContainer* dataContainer = cnode->GetNodeData();
        dataContainer->AddTaggedObject("Data", face);
        //dataContainer->AddString("AttributeType",
std::to_string(attributeType));
        dataContainer->AddInteger("AttributeType", attributeType);
        //dataContainer->AddString("value", value);
        attributes_tree_control0->InsertNode(cnode,      node,      afterNode,
attributes_tree_control0->NodeInsertOptionSort);
        cnode->ScrollTo(0,BlockStyler::Node::ScrollCenter);
        if(exp != NULL) { // add expression tag to the
            dataContainer->AddString("expressionTag",      std::to_string(exp-
>Tag()));
            //nodes_with_expression.insert(cnode->Tag());
            expressionToNodeMap.insert(make_pair(exp->Tag(), cnode->Tag()));
```

```
        }

        //dataContainer->AddString("expressionTag",  exp  ==  NULL?  "-1"  :
std::to_string(exp->Tag()));

        delete parentDatacontainer;
        delete dataContainer;
        return cnode;
    }
    else //create a new child node
    {
        parentNode                                                          =
dynamic_cast<BlockStyler::Node*>(NXObjectManager::Get(it->second));
        DataContainer* parentDatacontainer = parentNode->GetNodeData();
        auto   faces   =   parentDatacontainer->GetTaggedObjectVector("Data");
//property name is faces
        faces.push_back(face); // might not work becasue the vector is a
value not a reference
        parentDatacontainer->SetTaggedObjectVector("Data", faces);
        BlockStyler::Node*       node       =       attributes_tree_control0-
>CreateNode(value.c_str());
        DataContainer* dataContainer = node->GetNodeData();
        dataContainer->AddTaggedObject("Data", face);
        delete parentDatacontainer;
        delete dataContainer;
        attributes_tree_control0->InsertNode(node,   parentNode,   afterNode,
attributes_tree_control0->NodeInsertOptionSort);
        node->ScrollTo(0,BlockStyler::Node::ScrollCenter);
        return node;
    }
}

BlockStyler::Node*
FunctionalFacesVisualizer::CreateAssociatedFeatureNodes(Face*     face,     /*
function attribute note*/

BlockStyler::Node* afterNode) {
        clean_up_tree(*feature_tree_control01);
        auto part = dynamic_cast<Part*> (face->OwningPart());
        //print("the face is from part: " + part->Leaf());
        auto featureCollection = part->Features();
        //print("get feature collection for the part:" + part->Leaf());
        if(featureCollection == NULL) print("NULL feature collection");
        Features::Feature* feature;
        try {
            feature = featureCollection->GetAssociatedFeature(face);
            //feature = featureCollection->GetParentFeatureOfFace(face);
            //auto feature = featureCollection->GetParentFeatureOfFace(face);
            //print("get the feature: " + feature->GetFeatureName());
            if (feature == NULL)
            {
                //print("NULL feature from face data!");
                return afterNode;
            }
            //NXString nodename = feature->FeatureType()+ " " + " \" " +
feature->GetFeatureName() + "\"";
```

```
            BlockStyler::Node*      fnode      =      feature_tree_control01-
>CreateNode(feature->GetFeatureName());
            DataContainer * data = fnode->GetNodeData();
            data->AddTaggedObject("Data", feature);
            delete data;
            feature_tree_control01->InsertNode(fnode,     NULL,     afterNode,
feature_tree_control01->NodeInsertOptionLast);
            //std::string       iconString       =       std::string(feature-
>FeatureType().GetText());
            std::string      iconString      =      featureNameToType(feature-
>GetFeatureName().GetText());
            //replaceSpaceWithUnderline(iconString);
            fnode->SetDisplayIcon(iconString);
            fnode->SetSelectedIcon(iconString);
            //print("feature type:"+iconString);
            //print("feature name:" + feature->Name());
            if  (isAssembly)  fnode->SetColumnDisplayText(1,  part->Leaf());  //
only show the part name when it is an asembly
            //// add child and parent nodes
            if(isPopulateUp) {
                BlockStyler::Node*   parentNode   =   feature_tree_control01-
>CreateNode("Parents");
                feature_tree_control01->SetExpanded(true);
                feature_tree_control01->InsertNode(parentNode,  fnode,  NULL,
feature_tree_control01->NodeInsertOptionAlwaysFirst);
                parentNode->SetDisplayIcon("folder");
                parentNode->SetSelectedIcon("folder");
                DataContainer* pdatacontainer = parentNode->GetNodeData();
                pdatacontainer->AddTaggedObject("Data", NULL);
                delete pdatacontainer;
                PopulateFeatureTreeUp(feature, parentNode);
            }
            if (isPopulateDown)
            {
                BlockStyler::Node*   childrenNode   =   feature_tree_control01-
>CreateNode("Children");
                feature_tree_control01->InsertNode(childrenNode, fnode, NULL,
feature_tree_control01->NodeInsertOptionLast);
                DataContainer* cdatacontainer = childrenNode->GetNodeData();
                cdatacontainer->AddTaggedObject("Data", NULL);
                delete cdatacontainer;
                childrenNode->SetDisplayIcon("folder");
                childrenNode->SetSelectedIcon("folder");
                PopulateFeatureTreeDown(feature, childrenNode);
            }
            CreateAssociatedExpressionNodes(feature, NULL);
    }

    catch(exception &ex) {
        print("Exception  while  trying  to  create  feature  node  for  selected
face");
    }

    return afterNode;
}

void FunctionalFacesVisualizer::ProcessFace(Part* part, Face* face){
```

```
//  print("calling porcess face");

    auto attrs = face->GetUserAttributes(false); // dont't include unset
atributes
    if (attrs.empty()) return;

    for(auto it = attrs.begin(); it != attrs.end(); ++it)
    {
        std::string title = it->Title.getText();
        std::string strValue = std::string(it->StringValue.GetText());
        NXObject::AttributeType attributeType = it->Type;
        CreateAndAddFunctionAttributeNode(title,   strValue,   attributeType,
NULL, face, it->Expression);

    }
}

void
FunctionalFacesVisualizer::SearchFunctionalFacesInComponent_helper(Assemblies
::Component*  comp, std::set<tag_t> & visited_parts) {
        // process current component
    Part* part = dynamic_cast<Part*> (comp->Prototype()->OwningPart());
    //BasePart*  part =  (comp->Prototype());

    if(visited_parts.count(part->Tag()) == 0) { // not visited
        //print("insert  new  part  with  tag  =  "  +  std::to_string(  part-
>Tag()));

        visited_parts.insert(part->Tag());
        std::vector<Face*> faces = get_faces_in_part(part);
        if (faces.empty()) return;
        for(auto fit = faces.begin(); fit != faces.end(); ++fit)
        {
            ProcessFace(part, *fit);
        }
    }

}

void
FunctionalFacesVisualizer::SearchFunctionalFaces_helper(Assemblies::Component
*  comp, std::set<tag_t>& visited_parts){
//  print("calling Search functional faces helper");
    SearchFunctionalFacesInComponent_helper(comp, visited_parts);

    std::vector<Assemblies::Component*> childComps = comp->GetChildren();
    if (childComps.empty()) return;
    for(auto it = childComps.begin(); it != childComps.end(); ++it)
    {
        SearchFunctionalFaces_helper(*it, visited_parts);
    }

}

void FunctionalFacesVisualizer::SearchFunctionalFaces()
{
    Assemblies::ComponentAssembly*compAssy = workpart->ComponentAssembly();
```

```
    if (compAssy == NULL) {
        //print("not an assembly");
        return;
    }
    Assemblies::Component* root = compAssy->RootComponent();

    if(root == NULL) {
        print("not an assembly");
        isAssembly = false;
        feature_tree_control01->SetShowMultipleColumns(false);
        std::vector<Face*> faces = get_faces_in_part(workpart);
        if (faces.empty()) return;
        for(auto fit = faces.begin(); fit != faces.end(); ++fit)
        {
            ProcessFace(workpart, *fit);
        }

        return;
    }
    isAssembly = true;
    //// need to open the assembly fully.
    std::vector<NXOpen::Assemblies::ComponentAssembly::OpenComponentStatus>
openStatus1;
    compAssy-
>OpenComponents(NXOpen::Assemblies::ComponentAssembly::OpenOptionWholeAssembl
y, root->GetChildren(),openStatus1);

    std::set<tag_t> visited_parts;
    SearchFunctionalFaces_helper(root, visited_parts);
}
void
FunctionalFacesVisualizer::OnSelectCallback_AttributeTree(NXOpen::BlockStyler
::Tree *tree, NXOpen::BlockStyler::Node *node, int columnID, bool selected)
{
    clear_highlight();
    //clean_up_tree(*expressions_tree_control02);
    //clean_up_tree(*feature_tree_control01);
    std::stringstream column;
    column << columnID;

    std::string text = std::string("OnSelectCallback Invoked: Node \"") +
std::string(node->DisplayText().GetText()) + std::string("\" ") +
std::string(selected?"Selected":"Deselected") + std::string(selected?" at
column ":"") + std::string(selected?column.str():"");
    //print(text);
    clean_up_tree(*feature_tree_control01);
    clean_up_tree(*expressions_tree_control02);
    if(selected) {
        if(node->FirstChildNode() != NULL) {
            auto faces = node->GetNodeData()->GetTaggedObjectVector("Data");
            if (faces.empty()) return;
            for(auto it = faces.begin(); it != faces.end(); ++it)
            {
                (dynamic_cast<DisplayableObject*> (*it))->Highlight(); //
highlight all the faces
            }
        }
```

```
        else { // only create feature node when selecting one face
            auto    face    =    dynamic_cast<Face*>    (node->GetNodeData()-
>GetTaggedObject("Data"));
            face->Highlight();
            CreateAssociatedFeatureNodes(face, NULL);
        }
    }
    //else {
    //  clean_up_tree(*feature_tree_control01);
    //  clean_up_tree(*expressions_tree_control02);
    //}
}


BlockStyler::Node*
FunctionalFacesVisualizer::CreateAssociatedExpressionNodes(/*BlockStyler::Nod
e* featureNode,*/ Features::Feature* feature, BlockStyler::Node* afterNode)
{

    auto expressions = feature->GetExpressions();
    if (expressions.empty()) return afterNode ;
    for(auto it = expressions.begin(); it != expressions.end(); ++it) {
        std::string nodename = (*it)->GetDescriptor().GetText();
        ////print("expression descriptor:" + nodename);
        nodename = nodename.substr(0, nodename.length()-2);
        //std::string nodename = (*it)->Equation().GetText();
        BlockStyler::Node    *    expression    =    expressions_tree_control02-
>CreateNode(nodename);
        DataContainer* expData = expression->GetNodeData();
        expData->AddTaggedObject("Data", *it);
        delete expData;
        expressions_tree_control02->InsertNode(expression,     NULL,     NULL,
expressions_tree_control02->NodeInsertOptionSort);
        expression->SetDisplayIcon("equals");
        expression->SetSelectedIcon("equals");

        expression->SetColumnDisplayText(ExpressionColumn2,             (*it)-
>RightHandSide());

        expression->SetColumnDisplayText(ExpressionColumn3,             (*it)-
>Equation());

        expression->SetColumnDisplayText(ExpressionColumn4,             (*it)-
>Description());
        //print("Expression type = " + (*it)->Type());
    }
}

// should have named it UpdateAssociatedExpressionNode
BlockStyler::Node*
FunctionalFacesVisualizer::EditAssociatedExpressionNodes(BlockStyler::Node*
node, Expression* exp)
{
    node->SetColumnDisplayText(ExpressionColumn2, exp->RightHandSide());

    node->SetColumnDisplayText(ExpressionColumn3, exp->Equation());
```

```
    tag_t expTag = exp->Tag();
    int numOfTagsFound = expressionToNodeMap.count(expTag);
    if (numOfTagsFound > 0)
    {
        for(auto it = expressionToNodeMap.equal_range(expTag).first; it !=
expressionToNodeMap.equal_range(expTag).second; ++it) {
            BlockStyler::Node* attrNode = dynamic_cast<BlockStyler::Node*>
(NXObjectManager::Get(it->second));
            attrNode->SetColumnDisplayText(0, exp->RightHandSide());
        }
    }
    return node;

}

NXOpen::BlockStyler::Tree::ControlType
FunctionalFacesVisualizer::AskEditControlCallback_ExpressionTree(NXOpen::Bloc
kStyler::Tree *tree, NXOpen::BlockStyler::Node *node, int columnID) {
    NXOpen::BlockStyler::Tree::ControlType          AskEditControl          =
BlockStyler::Tree::ControlTypeNone;
    return AskEditControl;

}

NXOpen::BlockStyler::Tree::BeginLabelEditState
FunctionalFacesVisualizer::OnBeginLabelEditCallback_ExpressionTree(NXOpen::Bl
ockStyler::Tree *tree, NXOpen::BlockStyler::Node *node, int columnID)
{
        NXOpen::BlockStyler::Tree::BeginLabelEditState   OnBeginLabelEdit   =
BlockStyler::Tree::BeginLabelEditStateDisallow;
        OnBeginLabelEdit = BlockStyler::Tree::BeginLabelEditStateDisallow;
        Expression  *exp  =    dynamic_cast<Expression*>(node->GetNodeData()-
>GetTaggedObject("Data"));

        //print("calling begin label edit call back");
        if (columnID == 0)
        {
            //print("begin label edit on column 0 not supported");
            //OnEndLabelEditCallback_ExpressionTree(tree,   node,   2,   exp-
>RightHandSide());
            return OnBeginLabelEdit;
        }

        if(columnID == 1 || columnID == 2) {
            OnBeginLabelEdit = BlockStyler::Tree::BeginLabelEditStateAllow;
        }
        return OnBeginLabelEdit;

}

NXOpen::BlockStyler::Tree::EndLabelEditState
FunctionalFacesVisualizer::OnEndLabelEditCallback_ExpressionTree(NXOpen::Bloc
kStyler::Tree *tree, NXOpen::BlockStyler::Node *node, int columnID, NXString
editedText){
    //print("calling end label edit call back");
```

```
    NXOpen::BlockStyler::Tree::EndLabelEditState        OnEndLabelEdit        =
BlockStyler::Tree::EndLabelEditStateRejectText;
    try
    {
        Expression  *exp  =    dynamic_cast<Expression*>(node->GetNodeData()-
>GetTaggedObject("Data"));
        try {
            if(columnID == 1) {
                exp->SetRightHandSide(editedText);
                OnEndLabelEdit                                                =
BlockStyler::Tree::EndLabelEditStateAcceptText;
                NXOpen::Session::UndoMarkId markId6;
                markId6                        =                        theSession-
>SetUndoMark(NXOpen::Session::MarkVisibilityVisible, "Update");
                int     nErrs1     =     FunctionalFacesVisualizer::theSession-
>UpdateManager()->DoUpdate(markId6);
                EditAssociatedExpressionNodes(node, exp);
            //  node->SetColumnDisplayText(2,exp->Equation());
            }
            else if (columnID == 2) {
                std::string splitStr(editedText.GetText());
                size_t ind = splitStr.find("=");
                //print("ind="+std::to_string(ind));
                std::string renameStr;
                std::string value(splitStr);
                if (ind != splitStr.npos) {
                    //print("find = ");
                    renameStr = splitStr.substr(0, ind);
                    //print("renameStr:" +renameStr);
                    // rename the expression
                     trim(renameStr);
                    //print("trimmed renameStr:" +renameStr);
                    workpart->Expressions()->Rename(exp,  renameStr);  //  todo
the expression might not belong to the workpart
                    value = splitStr.substr(ind+1);
                    //print("value:" + value);
                }
                trim(value);
                    //print("trimed value:" + value);
                exp->SetRightHandSide(value);
                OnEndLabelEdit                                                =
BlockStyler::Tree::EndLabelEditStateAcceptText;
                NXOpen::Session::UndoMarkId markId6;
                markId6                        =                        theSession-
>SetUndoMark(NXOpen::Session::MarkVisibilityVisible, "Update");
                int     nErrs1     =     FunctionalFacesVisualizer::theSession-
>UpdateManager()->DoUpdate(markId6);
                EditAssociatedExpressionNodes(node, exp);
            }
        }
        catch (exception& ex) {
            OnEndLabelEdit = BlockStyler::Tree::EndLabelEditStateRejectText;
        }
    //node->SetColumnDisplayText(2, exp->Equation());

    }
    catch(exception& ex)
```

```
    {
        OnEndLabelEdit = BlockStyler::Tree::EndLabelEditStateRejectText;
        FunctionalFacesVisualizer::theUI->NXMessageBox()->Show("Block
Styler", NXOpen::NXMessageBox::DialogTypeError, ex.what());
    }
    return OnEndLabelEdit;
}


void       FunctionalFacesVisualizer::PopulateFeatureTreeUp(Features::Feature*
feature, BlockStyler::Node* parentNode)
{
    auto parents = feature->GetParents();
    for(auto it = parents.begin(); it != parents.end(); ++it)
    {
        if ((*it)->IsOccurrence() || (*it)->IsInternal()) continue;
        BlockStyler::Node*  node  =  feature_tree_control01->CreateNode((*it)-
>GetFeatureName());
        DataContainer* dataContainer = node->GetNodeData();
        dataContainer->AddTaggedObject("Data", *it);
        feature_tree_control01->InsertNode(node,       parentNode,       NULL,
feature_tree_control01->NodeInsertOptionSort);
        delete dataContainer;
        std::string        iconString        =        featureNameToType((*it)-
>GetFeatureName().GetText());
        node->SetDisplayIcon(iconString);
        node->SetSelectedIcon(iconString);
        PopulateFeatureTreeUp(*it, node);
    }

}

void     FunctionalFacesVisualizer::PopulateFeatureTreeDown(Features::Feature*
feature, BlockStyler::Node* parentNode)
{
    auto parents = feature->GetChildren();
    for(auto it = parents.begin(); it != parents.end(); ++it)
    {
        if ((*it)->IsOccurrence() || (*it)->IsInternal()) continue;
        BlockStyler::Node*  node  =  feature_tree_control01->CreateNode((*it)-
>GetFeatureName());
        DataContainer* dataContainer = node->GetNodeData();
        dataContainer->AddTaggedObject("Data", *it);
        feature_tree_control01->InsertNode(node,       parentNode,       NULL,
feature_tree_control01->NodeInsertOptionSort);
        delete dataContainer;

        std::string        iconString        =        featureNameToType((*it)-
>GetFeatureName().GetText());
        node->SetDisplayIcon(iconString);
        node->SetSelectedIcon(iconString);
        PopulateFeatureTreeDown(*it, node);
    }

}
```

```
void
FunctionalFacesVisualizer::OnSelectCallback_FeatureTree(NXOpen::BlockStyler::
Tree *tree, NXOpen::BlockStyler::Node *fnode, int columnID, bool selected)
{
    clean_up_tree(*expressions_tree_control02);
    if (selected) {
        try{
        DataContainer* data = fnode->GetNodeData();
        auto       feature       =       dynamic_cast<Features::Feature*>(data-
>GetTaggedObject("Data"));
        delete data;
        if (feature == NULL) return; // select the parent of children node,
no feature is associated to them and hence no expression
        CreateAssociatedExpressionNodes(feature, NULL);
        }
        catch (exception &ex) {
            std::string w(ex.what());

            print("Exception  while  selecting  a  feature  node:"  +fnode-
>DisplayText() + ". what = " + w);
        }
    }
    //else {
    //  clean_up_tree(*expressions_tree_control02);
    //}
}


enum AttributeMenuID
{
    DeleteNode = 0,

};

void
FunctionalFacesVisualizer::OnMenuCallback_AttributeTree(NXOpen::BlockStyler::
Tree *tree, NXOpen::BlockStyler::Node *node, int menuItemID)
{
    try
    {
        BlockStyler::TreeListMenu *menu = tree->CreateMenu();
        if (node == NULL) {
            return;
        }
        else
        {
            menu->AddMenuItem(DeleteNode, "Delete Node");
            menu->SetItemIcon(DeleteNode, "delete");
        }
        tree->SetMenu(menu);
        delete menu;

    }
    catch (exception &ex)
    {
        std::string w(ex.what());
```

```
        print("Exception from menu call back for the attribute tree. what = "
+ w);
    }

}

void
FunctionalFacesVisualizer::OnMenuSelectionCallback_AttributeTree(NXOpen::Bloc
kStyler::Tree *tree, NXOpen::BlockStyler::Node *node, int menuItemID)
{
    try
    {
        if (node == NULL) return;
        if ((AttributeMenuID) menuItemID == DeleteNode)
        {
            DeleteUserAttribute_AttributeTreeNode(node);
        }

    }
    catch(exception &ex)
    {
        std::string w(ex.what());

        print("Exception from menu selection call back for the attribute
tree. what = " + w);

    }

}

void
FunctionalFacesVisualizer::DeleteUserAttribute_AttributeTreeNode(BlockStyler:
:Node* node)
{
    clean_up_tree(*expressions_tree_control02);
    clean_up_tree(*feature_tree_control01);
    if (node == NULL) return;
    BlockStyler::Node* parent = node->ParentNode();
    if (parent == NULL)  // delete all the children nodes
    {
        BlockStyler::Node *cnode = node->FirstChildNode();
        while (cnode != NULL) {
            BlockStyler::Node* nextSiblingNode = cnode->NextSiblingNode();
            try{
                DataContainer* data =  cnode->GetNodeData();
                auto         face         =         dynamic_cast<Face*>(data-
>GetTaggedObject("Data"));

                NXOpen::AttributePropertiesBuilder
*attributePropertiesBuilder;
                Part* part = dynamic_cast<Part*>(face->OwningPart());
                std::vector<NXOpen::NXObject *> objects(1);
                objects[0]=face;

                attributePropertiesBuilder  =  theSession->AttributeManager()-
>CreateAttributePropertiesBuilder(part,                                  objects,
NXOpen::AttributePropertiesBuilder::OperationTypeNone);
```

```
                NXOpen::ObjectGeneralPropertiesBuilder
*objectGeneralPropertiesBuilder;
                objectGeneralPropertiesBuilder   =   part->PropertiesManager()-
>CreateObjectGeneralPropertiesBuilder(objects);
                //attributePropertiesBuilder->SetCategory();

                attributePropertiesBuilder->SetTitle(node->DisplayText());
                attributePropertiesBuilder->SetStringValue(cnode-
>DisplayText());
                attributePropertiesBuilder->Delete(face);

                attributePropertiesBuilder->SetCategory("");
                attributePropertiesBuilder->SetTitle("");
                attributePropertiesBuilder->SetStringValue("");
                NXOpen::Session::UndoMarkId markId42;
                markId42                          =                          theSession-
>SetUndoMark(NXOpen::Session::MarkVisibilityInvisible, "Face Properties");
                NXOpen::NXObject *nXObject;
                nXObject = attributePropertiesBuilder->Commit();
                NXOpen::NXObject *nXObject1;
                nXObject1 = objectGeneralPropertiesBuilder->Commit();
                int nErrs7;
                 NXOpen::Session::UndoMarkId id7;
                 id7                          =                          theSession-
>GetNewestUndoMark(NXOpen::Session::MarkVisibilityVisible);
                nErrs7 = theSession->UpdateManager()->DoUpdate(id7);
                objectGeneralPropertiesBuilder->Destroy();
                attributePropertiesBuilder->Destroy();
                attributes_tree_control0->DeleteNode(cnode);
                delete data;
            }
            catch(exception &ex) {
                std::string w(ex.what());

                print("Exception on deleting user attribute for a parent node
for the attribute tree. what = " + w);
            }
            cnode = nextSiblingNode;
        }
        attributes_tree_control0->DeleteNode(node);
    }
    else
    {
        try{
            DataContainer* data =  node->GetNodeData();
            auto face = dynamic_cast<Face*>(data->GetTaggedObject("Data"));
            delete data;

            NXOpen::AttributePropertiesBuilder *attributePropertiesBuilder;
            Part* part = dynamic_cast<Part*>(face->OwningPart());
            std::vector<NXOpen::NXObject *> objects(1);
            objects[0]=face;

            attributePropertiesBuilder    =    theSession->AttributeManager()-
>CreateAttributePropertiesBuilder(part,                                    objects,
NXOpen::AttributePropertiesBuilder::OperationTypeNone);
```

```
            NXOpen::ObjectGeneralPropertiesBuilder
*objectGeneralPropertiesBuilder;
            objectGeneralPropertiesBuilder    =    part->PropertiesManager()-
>CreateObjectGeneralPropertiesBuilder(objects);
            //attributePropertiesBuilder->SetCategory();

            attributePropertiesBuilder->SetTitle(parent->DisplayText());
            attributePropertiesBuilder->SetStringValue(node->DisplayText());
            attributePropertiesBuilder->Delete(face);

            attributePropertiesBuilder->SetCategory("");
            attributePropertiesBuilder->SetTitle("");
            attributePropertiesBuilder->SetStringValue("");
            NXOpen::Session::UndoMarkId markId42;
            markId42                          =                          theSession-
>SetUndoMark(NXOpen::Session::MarkVisibilityInvisible, "Face Properties");
            NXOpen::NXObject *nXObject;
            nXObject = attributePropertiesBuilder->Commit();
            NXOpen::NXObject *nXObject1;
            nXObject1 = objectGeneralPropertiesBuilder->Commit();
            int nErrs7;
             NXOpen::Session::UndoMarkId id7;
             id7                              =                          theSession-
>GetNewestUndoMark(NXOpen::Session::MarkVisibilityVisible);
            nErrs7 = theSession->UpdateManager()->DoUpdate(id7);
            objectGeneralPropertiesBuilder->Destroy();
            attributePropertiesBuilder->Destroy();

            attributes_tree_control0->DeleteNode(node);
            if  (parent->FirstChildNode()  ==  NULL)  attributes_tree_control0-
>DeleteNode(parent);
        }
        catch(exception &ex) {
            std::string w(ex.what());
            print("Exception  on  deleting  user  attribute  for  the  attribute
tree: what = " + w);
        }
    }

}
```