Machine learning in customer-driven product configuration based on lifecycle metrics

Huang, Youliang

2007

Masters Thesis

Machine Learning in Customer-Driven Product Configuration Based on Lifecycle Metrics

Huang Youliang

G0600867E

School of Computer Engineering

A thesis submitted to

Nanyang Technological University

in fulfillment of the requirement for the degree of

Master of Engineering

2007

ABSTRACT

Mass customization has become a major trend for the manufacturing sector today, as it enables companies to produce customized products that meet customers' individual requirements at mass production rate. Product configuration tool is an integral part of the manufacturing service system that has been recognized as the key enabler to achieve mass customization. Although intensively studied in the literature, the application of the product configuration is still in the initial stage, due to the lack of systematic modeling and efficient configuration approach. Heterogeneous product information in configuration has to be effectively managed. In this thesis, firstly we give an intensive study on the state-of-the-art computer-aided product configuration methodologies. The advantages and limitations of existing approaches are evaluated. Constraint-based approach is proposed to model and solve product configuration problem. It is based on Constraint Satisfaction Problem (CSP) paradigm, where problem is modeled into variables, domains and constraints. Based on the approach, a systematic product configuration system framework is then defined. Novel product data model and knowledge representation model are designed and developed to capture the domain problem. Moreover, to address the limitation of the manual knowledge acquisition approach for knowledge-based configuration system, a novel machine learning approach which deploys association

ABSTRACT

rule mining technique is proposed and implemented to achieve automatic product configuration knowledge generation. A product configuration software system is developed to implement the constraint-based configuration approach proposed, which demonstrates the significance of our modeling and reasoning approach. Constraint-based configuration is proven to be an effective and efficient approach. To further enhance our approach, the system is extended to take cost as design criteria. A cost model that provides cost assessment for whole product life cyle has been developed. A system module is developed and integrated with the configuration system to provide cost estimation capability, which is a significant value-add for practical product configuration application.

ACKNOWLEDGEMENTS

First and foremost, I would like thank my advisor Associate Professor Ng Wee Keong for his valuable guidance, consistent support and encouragement throughout the course of my graduate study at Nanyang Technological University in Singapore. From the start of my graduate course till the submission of this thesis, and Dr. Ng has been inspirational and is the role model of a hardworking and creative scientist, who has exhibited lots of passion in his work both in his teaching duties and research.

I would also like to thank Dr. Liu Haifeng for regular project meeting and valuable ideas. I am grateful to Associate Professor Lu Wenfeng from National University of Singapore in Singapore, and Dr Song Bin from Singapore Institute of Manufacturing Technology for their guidance and advice. I must also thank Associate Professor Lim Ee Peng and Dr Li Xiang for the suggestion and valuable advice during the project discussion sessions.

Lastly, I want to thank my friends and colleagues from CAIS, Phua Si Jie, Wang Lin, Han Shuguo for their generous help. Special thanks to my wife Chen Lihong for her continuous support during the whole Masters study.

TABLE OF CONTENTS

Table of Contents

LIST OF FIGURES

List of Figures

CHAPTER 1 INTRODUCTION

1.1 Background

In recent years, the business environment for the manufacturing industry has changed significantly. The global market becomes more and more competitive. The manufacturing companies are facing great on-going challenges. Many product-oriented organizations are being faced with the challenge of providing as much variety of products as possible for market to meet the customer's increasing personalized need [33, 54]. They often have to struggle with the trade-off among time-to-market, product variety and mass efficiency [47]. In order to be competitive, more and more manufacturers are seeking for strategies to produce customized products with near mass production efficiency. This trend has shifted the manufacturing paradigm from mass production to mass customization.

Mass customization as a competitive strategy is getting progressively increasing attention in both business and academic areas due to its high potential to provide sustained strategic advantage in a unique fashion [24]. From the state-of-the-art, mass customization can be defined either broadly or narrowly [43]. The broad concept has been focusing on the conceptual idea of the mass customization and how the parties involved work together to achieve it. The definition can be found from the following researchers' work. Piller [44] gives a comprehensive definition of mass customization as "customer co-design process of products and services, which meet the needs of individual customer with regard to certain product features. All operations are performed within a fixed solution space, characterized by stable but still flexible and responsive processes. As a result, the costs associated with customization allow for a price level that does not imply a switch in an upper market segment". Similar visionary views are provided by other researchers [15, 22, 44, 48] to illustrate the broad concept. Another similar, but narrower and more practical concept, which focuses on the technology aspect of achieving mass customization, are defined as "a system that uses information technology, flexible processes, and organizational structures to deliver a wide range of products and services that meet specific needs of individual customers, at a cost near that of mass produced items". In our research, we aim to develop computer-aided approach and information system to equip manufacturers with mass customization capability. Thus, our understanding of the definition is closer to the narrower concept as discussed.

In order to achieve mass customization, the information technology that allows products or services to be configured based on individual requirements in a fast and accurate fashion is most critical. The system based on this technology is known as product configurator [16], which has been recognized as key enabler for mass customization. It has been recognized as an important integral part of today's manufacturing services system to manage product variants and generate customized products that meet customers' personalized need [21]. Automatic product configuration using information technology can greatly enhance company's mass customization capability as it is fast and efficient. To be more specific, the core of configuration task is selecting and arranging combinations of component variants that satisfy given specifications on the basis of available product components and predefined rules for component composition. A number of previous work has dealt with IT-supported configuration for consumer products. Companies like Dell Computer and American Power Conversion (APC) have implemented their own web-based configurator to facilitate product sales [23]. It is useful for selecting highly modularized product components which are stored in company's database, where customization is not necessary. However, due to the limitation of the configuration technology, automatic configuration for most of the products in the market with complex composition is still not feasible and the configuration research is still in the early stage. Therefore, we are motivated by the need for a comprehensive product configuration approach and related issues in the current research.

On the other hand, although meeting requirement specification is the first priority of the configuration task, it is not sufficient for manufacturers to maintain customer satisfaction. In a business-to-business transaction, customers are concerned about three issues: *requirements*, *cost*, and *delivery*. Therefore, for a product configuration system, while it generates product variants that meets requirement specification, there would be a significant value-add to have the capability of cost estimation and manufacturing capability and lead time assessment. Therefore, product lifecycle cost and supply chain information are also key elements in a configuration system. In this research, we will also investigate the issues in the product lifecycle cost estimation and supply chain management. Appropriate technology and system are developed to integrate the key elements into the configuration system.

1.2 Objective

Although product configuration has been intensively studied in the state-of-the-art, the application is still in its initial stage and the effect is still limited. In order for manufacturers to achieve mass customization capability, to support customer-driven product configuration throughout the product's life cycle, and to enable rapid assessment and changes of product structure in response to customer requirement changes, this project will focus on research into the issues in the configuration problem and developing a computer-aided product configuration technology that supports personalized product generation. Product configuration system will be developed based on the technology to assist design engineers in generating more product variants. Product life cycle information such as life cycle cost information will be integrated with the configurator to provide designers and customers with more in-sight view of the product configurations.

The scope of work includes:

- A comprehensive literature review will be carried out to review the existing research technology contributed to the computer-aided product configuration. Different approaches will be assessed and new approach is to be proposed over the existing approaches;

- In order to model the configuration task, a product data model which is able to capture the necessary product information including product composition, component design information, and other lifecycle data has to be developed. It should be developed for data storage, exchange and configuration execution;

- Proper knowledge representation approach needs to be developed to capture design knowledge and provide fundamental support for configuration reasoning.

- A configuration system needs to be developed based on our configuration approaches.
- Research and investigate into the efficiency of the configuration system: Develop functions and capabilities for the configuration system to increase the value-add.

1.3 Organization of the Report

In Chapter 2, a comprehensive literature review on product configuration approaches is given. By comparing the various different approaches, Constraint-based product configuration is intensively studied and different extensions are presented. Issues of the current configuration research is discussed here. Overview of our configuration system is also presented. Chapter 3 shows the product family-based product data model. To enhance the representation power of the model, a cross-family product model is proposed. A constraint model is developed to represent configuration knowledge for constraint-based product configuration is shown and illustrated in Chapter 4. Based on the proposed approach, a constraint-based product configuration software system is designed and developed. It consists of several systems including Product Data Management, Constraint Manager, etc, is presented in Chapter 5. Lastly, Chapter 6 will conclude the work, list out the key contributions and bring up the issues for future work.

CHAPTER 2 LITERATURE REVIEW AND SYSTEM OVERVIEW

Representing and solving configuration problems have always been subjects of interest for applying and developing AI techniques. It requires powerful knowledge-representation models to capture the great variety and complexity of configurable product models. Moreover, efficient reasoning methods are necessary to provide intelligent interactive behavior in configurator software, such as solution search, satisfaction of user preferences, optimization, diagnosis, etc. Today, the number of configurable products available on the market is growing. The diversity and the complexity are expanding from conventional equipment configuration to software configuration, or service configuration, such as loans, insurance, travel packages, and so forth. Configuration is more than ever a challenging area for applying novel AI techniques since more and more sophisticated reasoning tasks are delegated to the configurator software. This review starts with the definition and issue of product congfugration. Various AI techniques will be presented and discussed.

2.1 What is "Product Configuration"?

A product configurator is an effective software tool to successfully implement the mass customization strategy [53]. Product configuration is a business process that supports choices for customers to find the best-suited product variants [6, 39]. A more comprehensive definition [11] is "software with logic capabilities to create, maintain and use electronic product models that allow complete definition of all possible product option and variation combinations, with a minimum of data entries and maintenance." And the core of a configuration task is selecting and arranging combinations of parts that satisfy the given specifications [39]. It displays two key features:

- The artifact being configured is assembled from instances of a fixed set of well-defined component types;
- Components interact with each other in predefined ways, i.e., interact according to the configuration and design rules.

Solving product configuration problem is like any other problem-solving activities. It is divided into two steps. First, the problem needs to be represented. In product configuration, the problem representation includes: (1) product data: the variation of the products need to be properly captured with a product model; (2) configuration knowledge: the predefined rules using which the components interact and are selected; (3) requirement specifications representation: the input from customers to indicate what final configuration results should be like. Second, an algorithm is needed to produce solutions based on the representation, i.e., the problem solver with given problem description and representation.

2.2 Knowledge-based Product Configuration

Most configuration systems [9] work through well-defined phases: user specifications lead to an abstract configuration, which then undergoes an expansion and refinement process until it yields a complete, detail configuration. The key configuration task is the domain knowledge modeling and reasoning. In this section, we present various knowledge-based configuration approaches that have been deployed in the past. The few popular approaches are shown in Figure 2.1, which will be briefly described and compared.

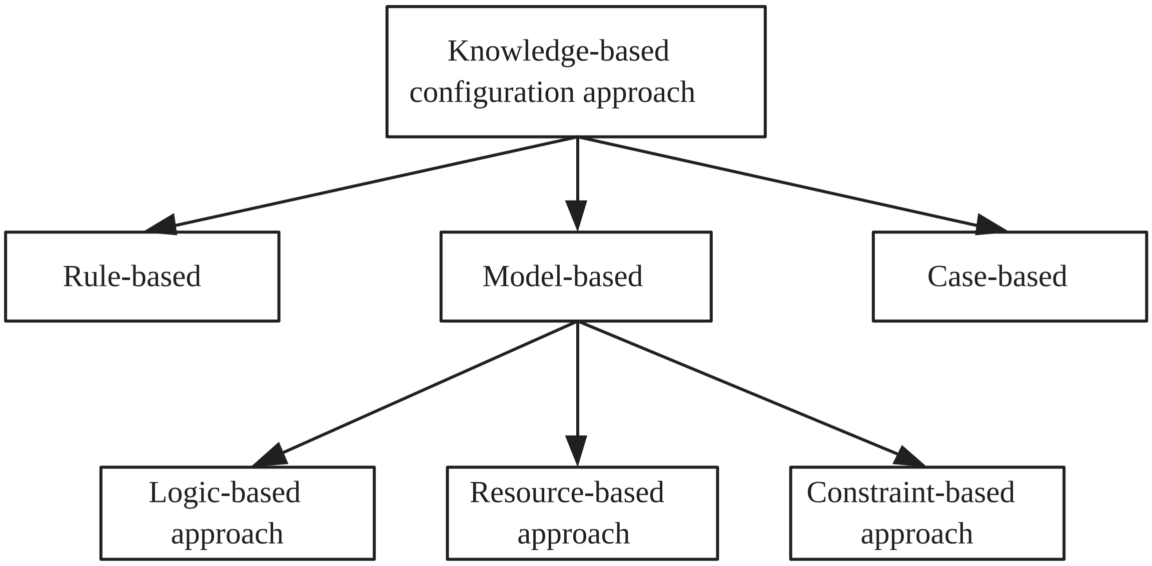


Figure 2.1: Knowledge-based Approaches for Product Configuration

2.2.1 Rule-based Reasoning

There are numerical configuration systems [9, 29] developed using rule-based reasoning. These systems are known as expert systems, which have the following features:

- Design knowledge is represented in specific rule language, which has its predefined syntax. e.g., JESS [1] and DROOL [4] are two common seen rule languages.
- Requirement specification acts as facts in the rule-based environment and is tested with the conditions part of the rules in the working memory, where rules are fired.
- Final configuration is produced when all rules are fired, and the firing process is executed by a rule engine, which has rule firing algorithm implemented.

Rule-based configurators use production rules as a uniform mechanism for representing both domain knowledge and control strategy and a working memory as a temporal information storage during the rule execution. The production rules has the form of *"if condition, then consequence"*. The working memory holds the information of inputs and results from executing actions. The conditional part of the production rule specifies tests on the working memory. If the condition is satisfied, the system executes the rule's consequence to bring the working memory to the new state. For example, $R1$ [29] is a typical rule-based configuration system. The system has a rule base that captures sufficient knowledge of the configuration domain and peculiarities of the various configuration constraints in each step of the configuration process. Consequently, a little search is required in order for it to configure a computer system. The rule-based system derives solutions in either a forward or backward chain manner. At each step, the system checks the entire set of rules and considers only the rules it can execute next. Each rule carries its own complete triggering condition. The system then selects and executes one of the rules under consideration by performing its action part.

In the early stages, rule-based configuration displays advantage in power of representing domain knowledge. The *if-then* structure makes the knowledge representation easy and intuitive. Existing company's design consideration and life cycle design criteria can be translated to production rules. However, as the configuration systems grow, the rule base becomes very large. This may incur enormous maintenance problem [9, 17]. Rules in the configuration system specify both domain knowledge, including compatibilities and dependencies, and rule execution control strategy. This lack of separation between domain knowledge and control strategy and the spread of knowledge about a single entity over several rules make the knowledge-maintenance task very difficult. In the case of updating certain rules, it would be very difficult to be certain that one has found all the rules that need changes.

2.2.2 Model-based Reasoning

Model-based reasoning, a subfield of AI research, was developed by researchers to address the limitation of the expert systems, which is primarily the maintenance problem. Model-based product configuration provides modeling facilities to enable the differentiation among three kinds of knowledge [52]:

- Conceptual knowledge includes concepts, taxonomic and compositional relations as well as restrictions between arbitrary concepts;
- Procedural knowledge declaratively describes the configuration process;
- A task specification specifies properties and constraints known from the customer that a product must fulfill.

There are three different types of model-based reasoning approaches, which are shown in the following section.

Logic-based approach The logic-based approach is based on description logic (DL) [7], which is formalism for representing and reasoning with knowledge.

In configuration, logic-based approach provides a clear framework to capture configuration semantics and simple logic operations to derive solutions. According to McGuinness and Borgida [30], the three fundamental notions of DLs are:

- *individuals*, representing objects in the domain;
- *concepts*, describing sets of individuals;
- *roles*, showing binary relations between individuals.

Composite descriptions are formed using *constructors*. Take the following representation as an example:

$$\text{RED\_WINE} \equiv (\mathit{and}(\mathbf{prim}\ \text{WINE})(\mathbf{fills}\ \mathit{color}\ \mathit{Red}))$$

In the above representation, RED_WINE is a *concept* that is defined using the **and** *constructor* to conjoin the primitive *concept* WINE with the description of *individuals* whose color *role* is filled by the *individual* RED. The main inference mechanism in DLs is *subsumption*, which is the decision whether one concept is more general than another. It is useful in integrating one concept into a concept hierarchy. Particularly, in product configuration, the product data is usually treated as a hierarchical structure. Thus, DL provides a good candidate for modeling the product data.

Logic-based configuration presents the advantage in clearly representing knowledge, which makes the knowledge acquisition and maintenance easy and efficient. Moreover, during the problem-solving phase, DLs enable the efficient retrieval of configuration constraints description, which is important add-on feature for configuration system that allows operators to control the configuration process. The limitation of the DLs approach lies in the tradeoff between expressiveness and reasoning efficiency. There is a level of the formalism expressiveness beyond which may it result in the situation that the reasoning efficiency become NP-complete.

Resource-based approach Resource-based configuration [18, 39] is another model-based approach that represents knowledge in different way. It treats the interface between elements including components and environment as resource. Each element is characterized by the amount and type of resources it supplies and consumes. The configuration tasks are basically the supply and consume of resources among the elements. The final solution is derived when the overall resources reach a balance statement. The visual representation is as shown in Figure 2.2. The arrows represent the resource supply and consume relationship. The resource transactions are among the components within a product, as well as between product and the environment. Dotted arrow shows that the resource is used by the entity. Based on the modeling, the reasoning starts with the resource demand from the environment which is the requirement specification. The resource type that is not balanced yet is selected and the components that can supply the corresponding resource are found. This process repeats as the resource demand and supply propagates. Backtracking is performed in case of dead end. The solution can be derived until all the resources are balanced, i.e., the amounts of supply and demand are equal. Resource-based configuration offers advantage in the case of function-component matching. With known functional requirement and the function of a component, the approach is able to match them efficiently. However, it loses its efficiency when considering the technical constraint during configuration process, which is necessary for most of the technical configuration system. Moreover, other than functional mapping, structural compatibility problem is difficult to solve using this approach.

Constraint-based approach Constraint-based product configuration is one of the most important model-based approaches which models a configuration task as a Constraint Satisfaction Problem (CSP) [32]. A CSP is formulated by a set of variables V, a domain D of possible values for each variable, and a set of constraints

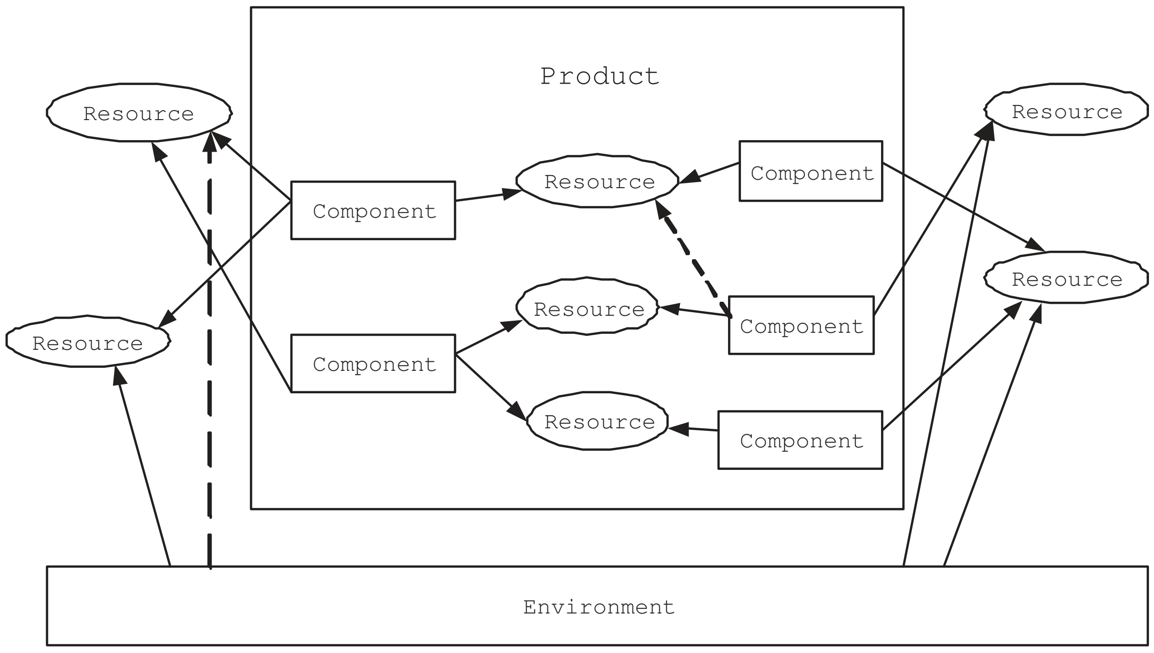


Figure 2.2: Resource-based Product Configuration: → *Supply*; ← *Consume*;

C defined to govern over the variables. A solution to a CSP is a complete assignment of values to the variables such that all constraints are simultaneously satisfied. Modeling configuration problem as CSP has the benefits of flexibility and generality. Product knowledge including requirement specifications, design consideration, and product data can be easily modeled as a CSP with concept of variables, domains and constraints. Thereafter, the algorithms can be solved independently of the product knowledge. Hence, the modeling and solving approaches are generic to the domain problem: product configuration problem. The detail formalism and solving algorithms of CSP will be further discussed in Section 2.4.

2.2.3 Case-based Reasoning (CBR)

Other than rule-based and model-based approach, case-based approach is another popular way of solving configuration problem. The principle of case-based product configuration is different from the other major configuration approaches. CBR assumes that the knowledge necessary for reasoning is stored mainly in historical cases that record a set of configurations that existed in the past. CBR solves a configuration by comparing the requirements with the historical data to find the similar transaction. Solutions will be retrieved based on similar requirements. Minor adaptation is needed to refine the similar solution to match with the current requirement. The major steps performed in the CBR system are:

- *Index assignment*: Assign index to the product features or components of the historical data in the database. The index will be used for identification during similarity measure and the case retrieval process;
- *Similarity measure*: The system compares the current customer requirements with the past transaction in existing database on the index assigned. Different similarity measure algorithms have been presented in the literature [36, 37]. Based on the algorithm, the most similar solution will be retrieved;
- *Case adaptation*: Adjust the retrieved case to fit the solution to the current status, which is usually done manually or with other AI approaches;
- *Case storage*: Update the database by storing the newly adapted case.

CBR had been successfully applied in several systems [13, 19, 27, 49] in the past. CBR addresses the issue of knowledge acquisition for traditional knowledge-based configuration system. However, the drawback that the method is not about to generate a complete solution as certain knowledge is difficult to discover from the cases. Therefore, using CBR in configuration problem alone is not optimized.

2.2.4 Summary

Each knowledge-based approach has its advantages and drawbacks. The comparison is shown in Table 2.1.

Table 2.1: Comparison among knowledge-based product configuration approaches

Approaches		Advantage	Limitation
Rule-based Reasoning		Representing and evaluating reasoning heuristic relations; Intuitive way of representing design knowledge	Lack of separation between knowledge and reasoning strategy, knowledge maintenance very difficult
Model-based Reasoning	Logic-based Approach	Clear semantic and simple logical operations	Reasoning efficiency trade-off with high expressiveness
	Resource-based Approach	Efficient in function-component matching	Unable to handle the internal technical consideration and constraints
	Constraint-based Approach	Domain independent, generic and flexible in modeling knowledge	Potential complexity in reasoning, i.e., consistency check; Completeness of configuration modeling is to be improved
Case-based Reasoning		Knowledge stored in existing cases, no need of knowledge acquisition phase	May result in inaccurate result; not suitable for new configuration

We are interested in the approach with sufficient robustness and accuracy in solving the domain problem. Rule-based approach has the drawback in maintenance problem, which is difficult to eliminate. Case-based approach may result in inaccuracy when the size of historical data is not sufficiently large. By comparing the various approaches [9, 7, 19, 32], we recognize constraint-based product configuration to be the best approach in terms of its generality and flexibility [45, 53]. Although its potential drawback is in the solving efficiency, the complexity can be reduced with proper problem modeling and efficiency solving algorithm. Despite the benefit of adapting CSP in configuration we can foresee, there are still challenges to be overcome. Novel value-addition to the system shown below is possible.

- Proper modeling is to be developed with our product data model to ensure soundness and completeness;
- Efficient knowledge acquisition process can be developed to enhance the acceptance of the system;

- Product life cycle information, such as lead time and cost, can be taken into consideration in order to enhance the competitive edge of the system;
- Strategies can be implemented so that more product recommendations can be generated to meet the diversified requirements.

2.3 Constraint Satisfaction Problem (CSP)

2.3.1 CSP Definition

Constraint Satisfaction Problem (CSP) has been recognized as a promising approach for modeling and solving product configuration problem [53]. Basically, a CSP is a problem composed of a finite set of variables, each of which is associated with a finite domain, and a set of constraints that restricts the values the variables can simultaneously take. The task is to assign a value to each variable satisfying all the constraints. Formally, they are defined as:

Definition 1. *The* ***domain of a variable*** *is a set of all possible values that can be assigned to the variable. If x is a variable, then we use D_x to denote the domain of it. A* ***label*** *is a variable-value pair that represents the assignment of the value to the variable. $\langle x, v\rangle$ is used to denote the label of assigning the value v to the variable x. $\langle x, v\rangle$ is meaningful only if v is in the domain of x (i.e. $v \in D_x$).* ***Compound label*** *can be represented as $(\langle x_1, v_1\rangle, \langle x_2, v_2\rangle, ..., \langle x_n, v_n\rangle)$, meaning that labels $v_1, v_2, ..., v_n$ are assigned to $x_1, x_2, ..., x_n$ respectively.*

Definition 2. *A* ***constraint*** *on a set of variables is a restriction on the values that they can take simultaneously. Conceptually, a constraint can be seen as a set that contains all the legal compound labels for the subject variables; The constraint on the set of variables x is denoted as C_x.*

Definition 3. *If the variables of the compound label X are the same as those variables of the elements of the compound labels in constraint C, then X satisfies C if and only if X is an element of C. Mathematically it is represented as:*

$$satisfies((\langle x_1, v_1\rangle\langle x_2, v_2\rangle...\langle x_k, v_k\rangle)|C_{x_1,x_2,...,x_k}) \equiv$$
$$(\langle x_1, v_1\rangle\langle x_2, v_2\rangle...\langle x_k, v_k\rangle) \in C_{x_1,x_2,...,x_k}$$

Definition 4. *A* ***Constraint Satisfaction Problem*** *is a triplet*

$$(Z, D, C)$$

where Z = finite set of ***variables*** $\{x_1, x_2, ..., x_n\}$,
D = a set of ***domain*** $\{D_{x_1}, D_{x_2}, ..., D_{x_n}\}$ *that maps with the* ***variables*** *in Z;*
C = finite set of ***constraints*** *on an arbitrary subset of variables in Z.*

2.3.2 CSP Solving Algorithms

CSP can be solved using the generate-and-test method, in which each possible combination of the variables is systematically generated and then tested to see if it satisfies all the constraints. The number of combinations considered by this method is the size of the Cartesian product of all the variable domains, which makes this method inefficient. Another more efficient way is to treat CSP as a search problem and use the backtracking method [41, 55]. Variables are instantiated sequentially in this method and the validity of the constraint is checked. If the instantiation violates any of the constraints, backtracking is performed to the most recently instantiated variable that still has alternative values. Although better than generate-and-test, the complexity for backtracking is still exponential, mainly due to its *thrashing*. The *thrashing* is caused by node inconsistency or arc inconsistency which results in repetitive backtracking. An arc (x, y) in the constraint graph of a CSP (Z, D, C) is arc-consistent if and only if for every value a in the domain of x which satisfies the constraint on x, there exists a value in the domain of y which is compatible with $\langle x, a \rangle$. In CSP research, various arc-consistency checking algorithms [10, 12] have been developed to enhance the complexity. Another factor that affects the practical CSP solving efficiency is the variable-ordering, which is known to have a potentially profound effect on its efficiency, and various heuristics [8] have been investigated for choosing good orderings. Therefore, constraint-based product configuration can adopt the existing solving algorithms with proper modeling domain problem and ordering of variables for the particular application.

2.4 Constraint-based Product Configuration

The attempt of applying a CSP to a configuration problem was first presented by Mittal and Frayman [32]. In their approach, a product was defined as a composition of a fixed, pre-defined set of components. Each components is characterized by a set of properties and ports for connecting to other components. The properties and ports are represented as variables with discrete and finite domains, and the restrictions on how the various components can be combined to form a valid configuration are represented as *compatibility constraints*. A configuration task is to assign values to all the variables without violating any constraint. They have shown that constraint-based approach has the advantage of a simple and domain-independent representation for a configuration problem. However, the assumption of a fixed and pre-defined set of components types make it inadequate to address more complex configuration problems. To address this issue, Dynamic Constraint-based Configuration (DCSP) [40], which is an extension to the classical CSP was proposed. In this approach, *Activity Constraint* is introduced to specify conditions under which a variable may or may not be actively considered as a part of a final solution. With DCSP, a product configuration can dynamically include or exclude a component type depending on the selections. The model of DCSP is represented below.

A Dynamic CSP $\mathbf{P} \equiv \{V,\ D,\ V_i,\ C_c,\ C_a\}$

where:

- $V\{V_1, V_2, ..., V_n\}$: A set of variables;
- $D\{D_1, D_2, ..., D_n\}$: Variable Domains;
- $V_i\{V_{i_1}, V_{i_2}, ..., V_{i_n}\}$: A set of initial variables, $V_i \subseteq V$;
- $C_c\{C_{c_1}, C_{c_2}, ..., C_{c_n}\}$: A set of compatibility constraints
- $C_a\{C_{a_1}, C_{a_2}, ..., C_{c_n}\}$: A set of activity constraints

The variables which are not *Initial Variables* is optional and may not assign a value in the final solution if it is not active. A variable can be activated by the *activity constraint*. Compatibility constraint C_c is active when all the variables it constrains is active.

Generative CSP [45] is proposed to further extend the DCSP models to make it more generic by specifying constraint among components in a meta-level instead of component instances. Generative CSP allows variables to have infinite domain values, which enhances the expressiveness. However, these extensions are still limited to model variables with discrete domains. In practical product configurations, continuous and numerical variables with a range of possible values are usually encountered, especially for the component parameters. Aldanondo et al. [5] proposed the requirements for modeling continuous variables and numerical constraints over both discrete and continuous variables. They pointed out that very few commercial configurators supported discrete and continuous variables. Based on the analysis, Xie et al. [53] proposed their constraint model, in which three types of constraint are defined: *compatibility constraint*, *inequality constraint*, and *equality constraint*. The *inequality* and *equality* constraints are represented intentionally by mathematical expressions or computable procedures that indicate a valid or invalid assignment for a consistency check. They also introduce *dependent variables* and *independent variables* to reflect the dependent relationship among different variables during the assignment, which in turn provides support for the variable ordering that affects the solving efficiency. To enhance systematic modeling of configuration problem, Sabin and Freuder [38] proposed a composite constraint satisfaction approach. In the approach, value for variables can be an entire set of subproblem. For *problem* $P \equiv \langle V, D_v, C_v \rangle$, instead of V taking atomic values, it can take a value representing an entire *subproblem* $P' \equiv \langle V', {D_v}', {C_v}' \rangle$. The effect of instantiating a variable V_p to the value of P' is that the problem we have to solve is dynamically modified and becomes $P'' \equiv \langle V'', D''_v, C''_v \rangle$, where:

- $V'' = V \cup V' - \{V_p\}$,
- $D''_v = D'_v \cup \{D_X | X \in V, X \neq U\}$,
- $C''_v = C_v \cup C'_v \cup C_{v',v} - C_{\{U\}}$

$C_{\{U\}}$ is the set of all constraints between variable U and other variable in V, and $C_{v',v}$ is the set of all constraints connecting a variable in V' to a variable in V. The constraints in $C_{v',v}$ can be seen as a refinement of the constraints in $C_{\{U\}}$ in the sense that previous restrictions imposed on U, by its neighbors in the constraint graph, become now more precise, referring explicitly to variables in U's internal structure.

This formulation enables the variables to have the hierarchical structure, which matches with the data structure of the majority of product model. This approach offers an increased representational power and efficiency for configuration problem. However, it only considers the compatibility constraints with discrete domain variables. Taking other constraints type and continuous variables in this approach may result in high complexity. From the classical CSP to the various CSP extensions, the expressive power and efficiency in modeling configuration problem have been improved. Although CSP has been applied to solve configuration problem, its modeling is still under development for optimal representation power and efficiency.

2.5 Configuration in Concurrent Engineering

In concurrent engineering [31], product design and configuration phase have been known to have great influence on the down stream activities, especially affecting the cost that would be incurred in the down stream life cycle phases, including material, manufacturing, assembly and testing cost. Weustink et al. [50] claimed that cost estimation before configuration process is critical and they proposed a framework to estimate cost in design, process planning and production in the early life cycle stage. Shehab and Abdalla [42] developed an intelligent knowledge-based system which is capabable of selecting a material, as well as machining processes and parameters based on a set of design and production parameters, and of estimating the product cost throughout the entire product development cycle including assembly cost. This work has significant contribution to the cost estimation. However, separating the configuration process with cost estimation suppresses the benefit of the cost estimation technology. By integrating the cost estimation capability into the configuration process, cost becomes one of the configuration criteria which support the designers' decision towards producing a product with an optimal quality and cost combination, i.e., better quality and lower cost.

2.6 Configuration System Overview

With the existing research work as fundamental support, we have proposed our configuration system framework. Figure 2.3 shows the overall system framework of the product configuration system. The bottom layer shows the product data model capturing necessary product lifecycle data, which provides a fundamental support for the product configuration and lifecycle cost assessment. In the data modeling layer, data model is classified to be used in different sub-functional modules. Here, we focus on the Original Equipment Manufacturing (OEM) sector instead of consumer products, where requirements from customers are usually stated in technical terms. Therefore, the configuration module will directly take the requirement specification from customers. With the data model and theory support, product configurator and lifecycle estimation module will be implemented and the interface will be developed for user interaction and system input/output. In this thesis, detail modeling and solving approach for the configuration module will be shown and discussed.

In the configuration module, one of the important task is to represent a product and its components. A product has to be represented in a product architecture in which the fundamental elements are arranged into physical units and the way in which these units interact. As discussed in the literature, configuration problem can be modeled as a generic CSP problem. Elements in configuration problem can be modeled and mapped into the CSP model. As shown in Figure 2.4, product data are modeled as variables in CSP. The possible values for the product components and component parameters are modeled as variable domains. Design knowledge and requirement specification indicate the restrictions on the selection of the component value which are modeled as constraints in CSP. With proper modeling of the variables and constraints, the configuration task can then be treated as a CSP solving process.

2.6 Configuration System Overview

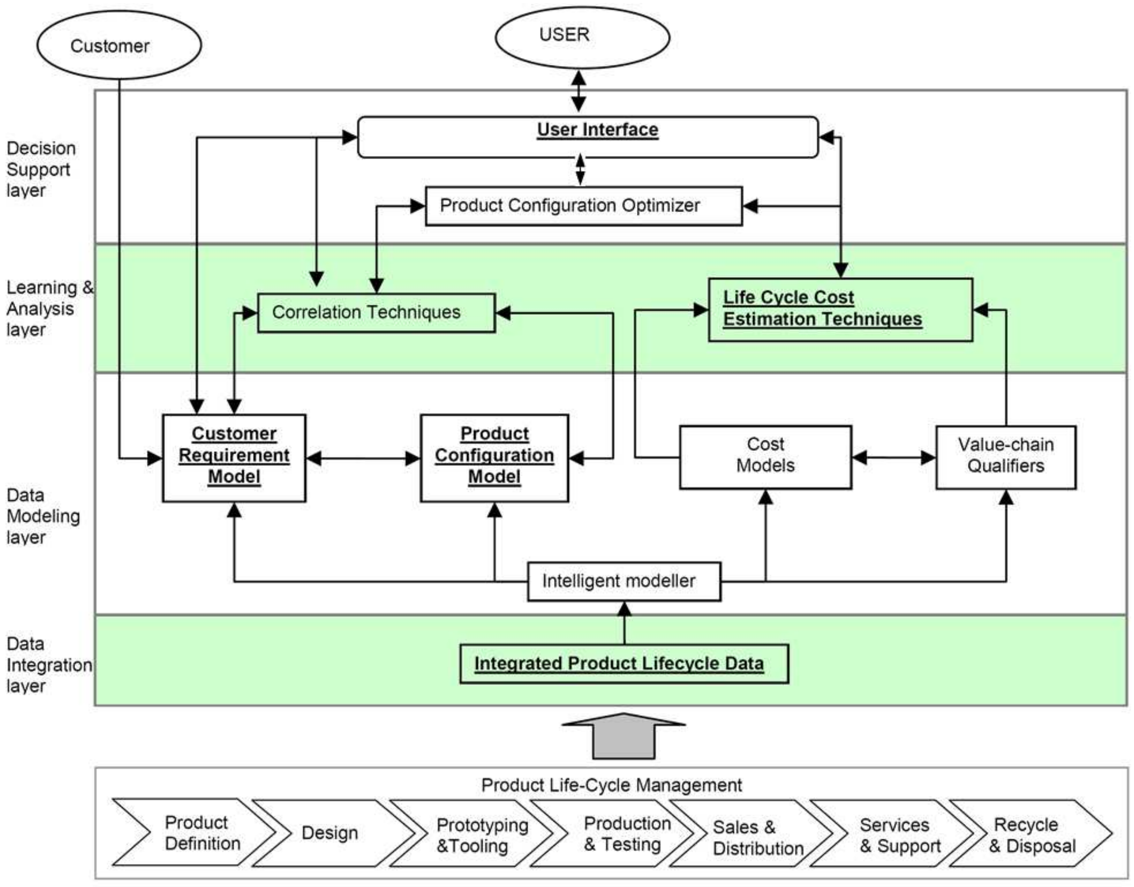


Figure 2.3: IMSS Manufacturing Service System Framework

Figure 2.5 shows the flowchart of the constraint-based product configuration. From the figure, we can see that the key elements in the configuration system includes product data model, constraint model, and the constraint solver. Product data model provides fundamental support to the system, which is required to capture the product design data, as well as other related life cycle data into computational model. As the number of product variants in mass customization is large, it is critical for the product data model to be able to efficiently manage the variety. Based on the product data model, constraint model acts as templates/schema capturing the semantics of requirement specification and design knowledge. CSP algorithm will be implemented in the CSP solver to generate product variants based on the product data

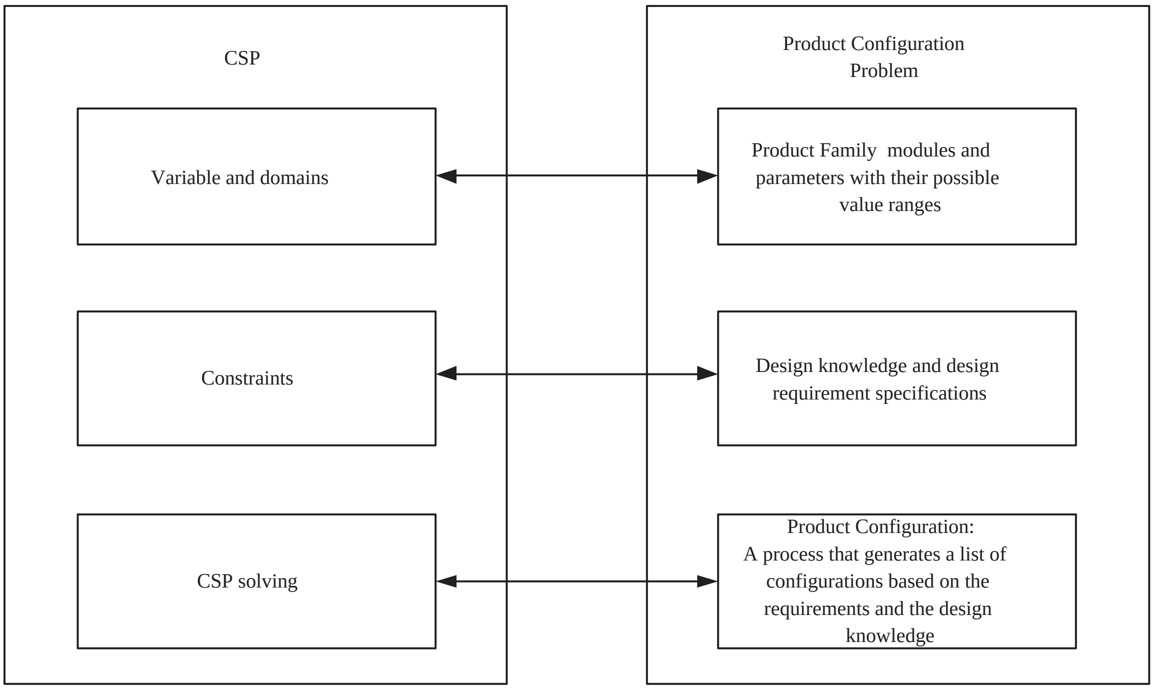

CSP
Variable and domains
Constraints
CSP solving
Product Configuration Problem
Product Family modules and parameters with their possible value ranges
Design knowledge and design requirement specifications
Product Configuration: A process that generates a list of configurations based on the requirements and the design knowledge

Figure 2.4: Constraint Satisfaction Problem and Configuration Problem

and constraints. The three key elements will be discussed in detail in the following sections.

2.7 Summary

Configuration problems have been defined and presented in the literature. As computer-aided product configuration can significantly enhance the efficiency in solving configuration problem, it has been a topic of interest in knowledge-based research. Researchers have been using different knowledge-based approach to model the problem, including three major ones presented: Rule-based, Model-based and Case-based approach. These approaches have been compared and constraint-based configuration, one of the model-based approach, have been adopted in this work. We review the general solving algorithms and knowledge modeling of constraint-based configuration. The significance from the past researchers has been shown.

2.7 Summary

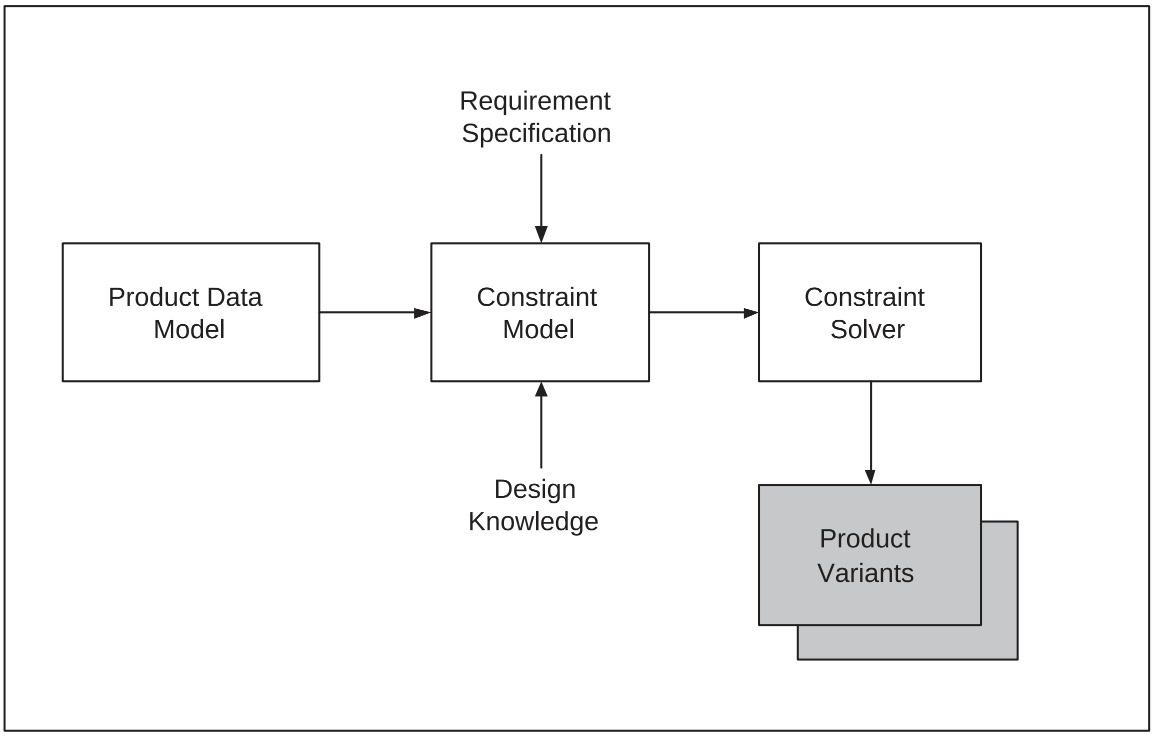


Figure 2.5: Configuration System Flowchart

However, traditional constraint-based Product Configuration can be enhanced by systematic modeling of the domain problem, including product and constraint modeling, variables and constraint ordering. Moreover, knowledge acquisition process in traditional systems is usually done manually, which is inefficient and error-prone. Efficiency can be enhanced by incorporating automatic knowledge acquision process into product configuration system. On the other hand, the manufacturing environment today become very competitive. Companies also compete with each other on lead time and cost. Therefore, life cycle information has been taken into consideration in the design and configuration phase as it is cost sensitive. The life cycle cost information becomes one of the criteria in the configuration decision making, which would significantly enhance the value of the traditional constraint-based product configuration. In this chapter, we also present a overview on our product configuration framework. Detail discussion on each individual module will be presented in the following chapters.

CHAPTER 3 PRODUCT DATA MANAGEMENT

3.1 Introduction

In product lifecycle management, proper data modeling is critical to integrate the information from different stages of lifecycle for design consideration. However, the challenge in capturing product data lies in the huge variety of the product data. Therefore, a comprehensive modeling concept must be introduced. In this chapter, we present a product-family modeling paradigm, product family architecture, to capture the hierarchical nature of product data and to cluster products into different sectors to ease the data management. Cross-family product data management is also proposed to enhance the features of the family-based data model.

3.2 Product Family Architecture

A Product Family Architecture (PFA) [14] consists of modules. A module is a physical or conceptual grouping of components. Modularity is the concept of decomposing a system into independent parts or modules that can be treated as logical units. In product modeling, the PFA has been recognized as an effective means of organizing product variations. A product family is defined as a set of individual products that share a common technology and address a related set of market segments. These individual products are commonly referred as product instances of the product family. PFA provides a simple and efficient hierarchical structure to model a product family. Figure 3.1 shows a PFA of a desktop PC and its product instances. The building block in PFA displays general functionally it provides and the one in product instance displays the structural and physical components.

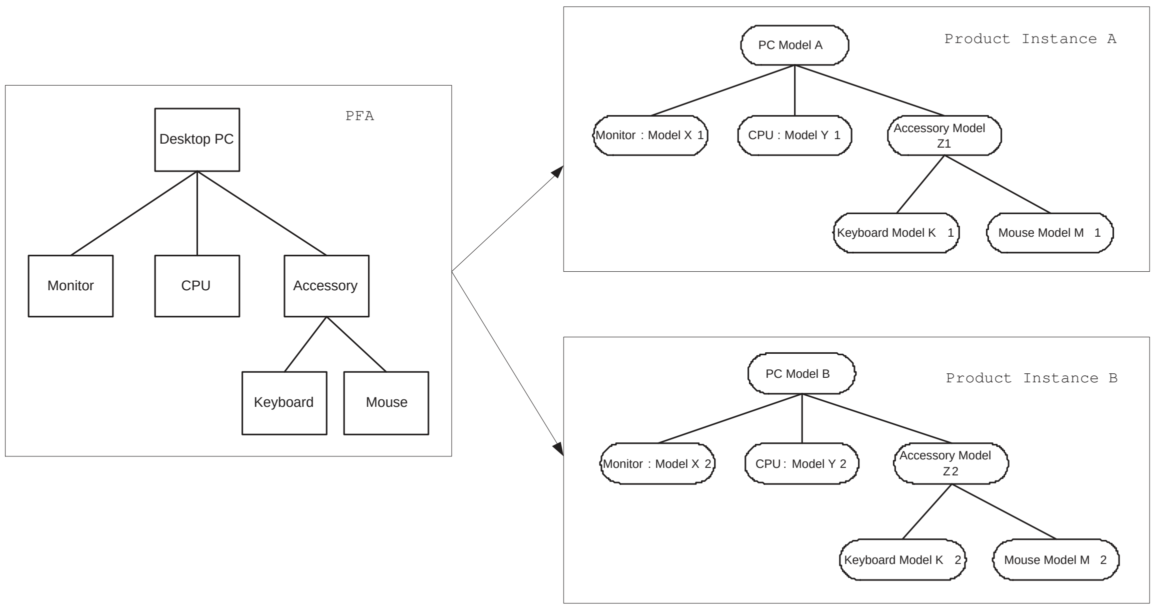


Figure 3.1: Product Family Architecture (PFA) and Product Instances

Modularity of PFA The concepts of modules and modularity are central in constructing a PFA. A product family architecture consists of a set of modules which are the conceptual grouping of product components. The modularity separates a system into independent parts or modules that can be treated as logical units, which makes the configuration task possible. Each module in the structural view has features showing the function that it provides and make the matching with functional requirements possible. Moreover, high level functional requirements can be decomposed to match with individual modules. The modularity also reduces the interaction between modules. Modules are defined in such a way that between modules interactions are minimized whereas within module-interaction is maximized.

Commonality versus Variety In PFA, modules are classified into two main categories, namely common modules and differentiation modules. *Common modules* are the elements that are shared by all the products across a product family. These modules in different products display common features from the functional view and they consist of common product structure and components. *Differentiation module* are the basic elements making a product within a product family different from another product. They are the source of variety for a product family. The differentiation modules are determined during the configuration according to the customers' requirements.

Scale-based PFA In a product family, the common modules are not necessarily identical. To allow generation of more product variants within a product family, the parameters in the common modules can be scaled according to the requirement specification, provided that the features of the common modules do not change.

3.3 Product Instance Model

The product family concept is used to model product data. Figure 3.2 shows the Entity Relationship Diagram (ERD) of the product model. A product family model which consists of a list of compound/primitive modules can be instantiated to generate a list of product variants by assigning values to parameters and/or selecting component alternatives. Based on this product model, a product data management (PDM) interface shown in Figure 3.2 is implemented to facilitate the product variants management, as well as component and parameters design and editing. The product model module is considered as a fundamental support for the configurator.

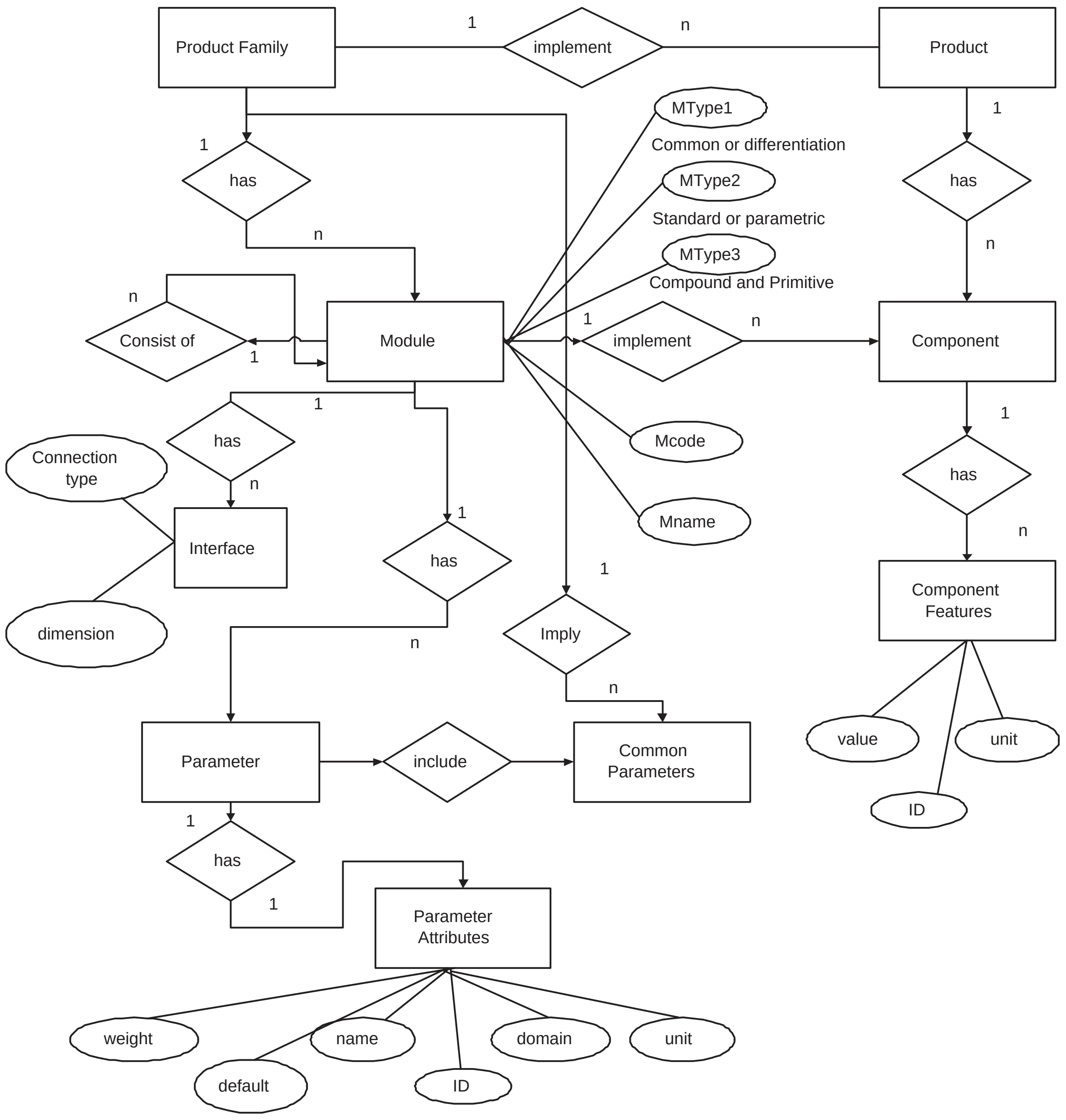


Figure 3.2: Product Data Model

There are two main threads in our Product Data Model (PDM), namely *Product Family* and *Product*. Entities are organized within the two main scopes and their internal relationships are captured. The key elements are listed and illustrated as follows.

3.3 Product Instance Model

Product Family The entity *product family* is the top level element that consists of a list of *modules*. The *module* are the main constitutions of a *product family*. It has three different attributes called *module type*. The *module type* can be:

- **Common Module** or **Differentiation Module**: These refer to the common and variety of the product family as discussed earlier. Common module is commonly shared across the product family and it is mandatory during the product generation, whereas differentiation module is the key enabler for product variety and it is optional during the product variant generation.
- **Compound Module** or **Primitive Module**: Primitive module is the lowest level element in a product family structure. Compound module consists of at least two sub-modules. It corresponds to the assembly in the product structure.
- **Standard Module** or **Parametric Module**: Standard module has industry standard parts. Its corresponding product component can be retrieved from the existing product database. They are usually associated with a standard part number. Parametric module requires customization and parameter setting.

The entity *Module* has a relational link to itself to form a nested module structure, which allows to capture the hierarchical structure of PFA. Each *Module* has an *Interface*, which provides the requirements or constraints for different module to assemble them together.

Entity *Parameter* is associated with *Module*. Each *Module* can have more than one *Parameter*, which is used to capture different attributes of *Module*. For example, a *Parameter* can be “Color”, “Weight”, etc. Each *Parameter* entity has attributes including, domain, value, unit, etc.

Product Entity *Product* in the PDM is an instantiation of *Product Family*. It consists of multiple *Component*, which is the instance of *Module*. Corresponding to the module structure, *Component* also has a nested structure and there are compound components and primitive components. Each *Component* has attributes including value, unit, etc, as its properties. When a configuration is complete, the attributes will be filled with the values so that the product configuration matches the requirements.

The mathematical model of the PDM: *Module* is the key element in the PDM that provides mapping relationship between product's functional view and structural view. Each module corresponds to a set of components. Variations of final product configuration can be managed by assigning different components to the modules. As product family is designed to meet customer requirements, the product family model must not only capture structural information, it should also represent the functional view of modules, as the requirements are usually specified as functions of the building blocks. Therefore, the notation of the module must take into account these issues. We use the following formalism to represent modules:

$$Module : (NS, NF, \overrightarrow{NV}, \overrightarrow{A}, \overrightarrow{CN})$$

- NS : Structural view of a node;
- NF : Functional view of a node;
- $\overrightarrow{NV}$: Node variants: A vector that captures all the varieties of instances of a module;
- $\overrightarrow{A}$: Attributes list of a module;
- $\overrightarrow{CN}$: Child nodes of a module.

The product data model is implemented into MS SQL database. It is also represented in the XML document. An example of XML document is shown in Figure 3.3. It is used for data exchange and integration with other systems. The product data management system is developed based on the model. It allows user to manipulate and populate module data through interface.

```
<Module >
      <NS>Monitor </NS>              <!--node's structural notation  -->
      <NF>Display </NF>               <!--node's fucntional notation  -->
      <ATTLIST >                      <!--attribute list of the node  -->
            <ATT1 >Size </ATT1 >       <!-- attirbute label of the node  -->
            <ATT2 >Type </ATT2 >
            <ATT3 >Color </ATT3 >
      </ATTLIST >
      <CNLIST >                       <!--child node list of the node  -->
            <CN1 >Frame </CN1 >       <!-- child node label  -->
            <CN2 >Circuits </CN2 >
      </CNLIST >
      <NVLIST >                        <!-- node variants list  -->
            <NV1 >Model 221 </NV1 >   <!--one node variant label  -->
            <NV2 >Model 312 </NV2 >
      </NVLIST >
</Module >
```

Figure 3.3: XML representation of module“monitor” in computer configuration

3.4 Cross-Family Product Configuration

In product family architecture, products must inherit the common structure and features of their corresponding product families. To a certain degree, PFA constrains the variations of the product instances. How to generate more variety to meet customers’ personalized needs with minimum resources is a challenge faced by product-oriented organizations [34]. In order to address this issue, we introduce the concept of *cross family* product configuration. Unlike traditional product family based configuration, where a product instance is configured based on a singular predefined PFA, cross family product configuration adds feature to allow product instance to be generated from two or more PFAs. The product instance inherits a portion of features from the PFAs that it is derived. In cross family product configuration, requirements specification is generally matched by more than one PFAs. In order to enable on-the-fly cross family product configuration, it is critical to rapidly extract the semantics content, which are the rationales of the configuration, from the existing source. Therefore, product family semantics content should be properly modeled and a methodology must be proposed to search for semantics in the existing product families and extract it for configuration.

3.4.1 Cross-Family Configuration Framework

Figure 3.4 shows the framework for integrating our product semantic model into traditional product family-based configuration. Customer requirements are captured by the specification model, which is a hierarchical structure that consists of functional modules and features for a particular product family. The knowledge in the specification model is derived from customer requirements analysis. The semantic model is designed to capture the semantic content of the knowledge for product configuration. It consists of the PFA model and the corresponding constraint model. The requirements that are properly represented in the specification model can be quickly matched to its corresponding product family. The semantic model for product family can be retrieved and used by the CSP solver. A set of final product variants that satisfy the constraints can then be generated.

However, it turns out that in some cases, the specification model does not entirely match any particular PFA model. This is the case when cross family product configuration comes into play. Figure 3.5 shows the framework of the cross family product configuration. The similarity between the requirements specification and the available product family features are compared and indexed. A novel algorithm for tree similarity measure is proposed and illustrated in Section 3.4.3. Using the algorithm, related product family compositions will be identified and the corresponding constraints can be extracted and integrated. Post-processing is carried out by a configuration knowledge engineer to check for completeness of the constraints. The completed semantic information can then be used by the CSP solver to generate a set of final product configurations that meet the requirements specification.

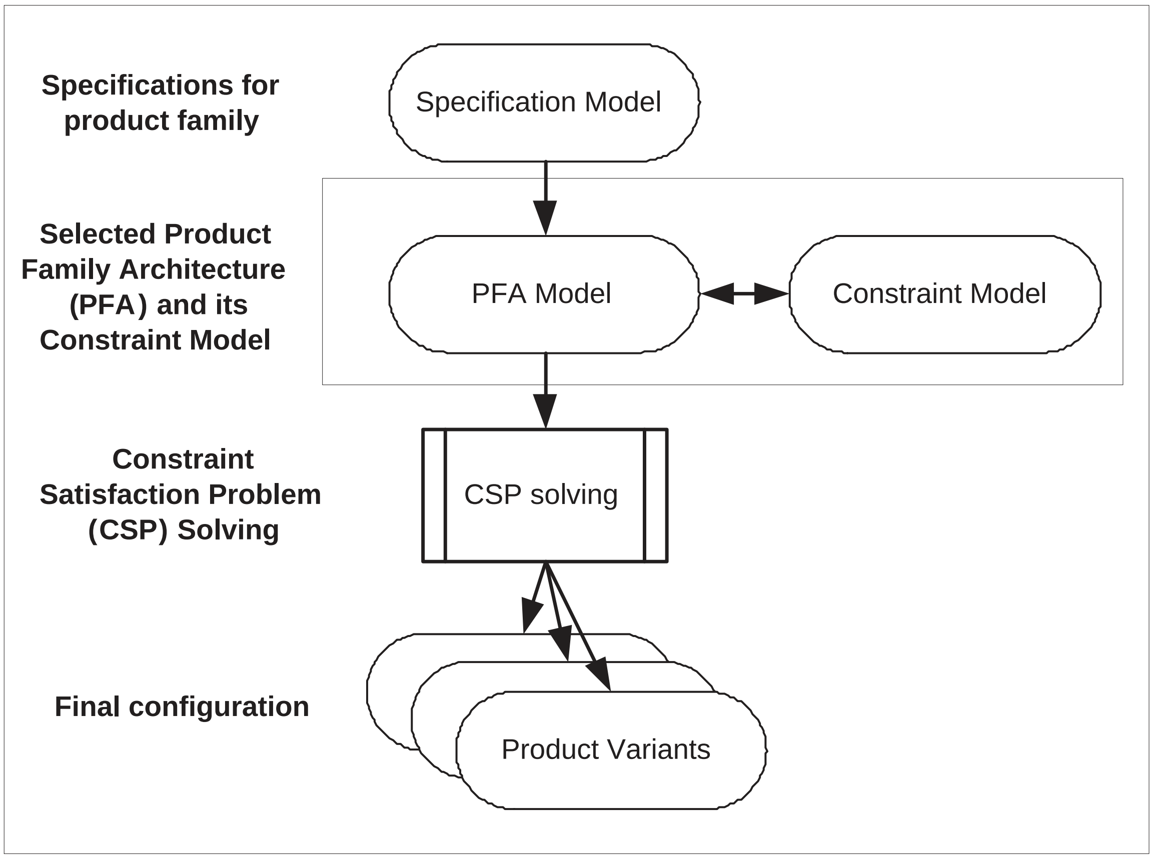

Specifications for product family
Specification Model
Selected Product Family Architecture (PFA) and its Constraint Model
PFA Model
Constraint Model
Constraint Satisfaction Problem (CSP) Solving
CSP solving
Final configuration
Product Variants

Figure 3.4: Product Family-based Configuration

3.4.2 Semantic Model for Product Family

The semantic model is a core component in the whole configuration process. It provides the linkage among different system modules, including the requirements specification and the CSP solver. Therefore, it should capture semantic contents in different aspects. As shown in Figure 3.6, the model consists of two parts: the PFA model and the constraint model. A product family is represented using a tree structure that shows the composition of the product family as well as the structural relationships among the components. The constraints are represented in the graph-based model as shown. It can be seen visually that the constraint model is a projection of the product family tree onto the ground. The model demonstrates the concept of

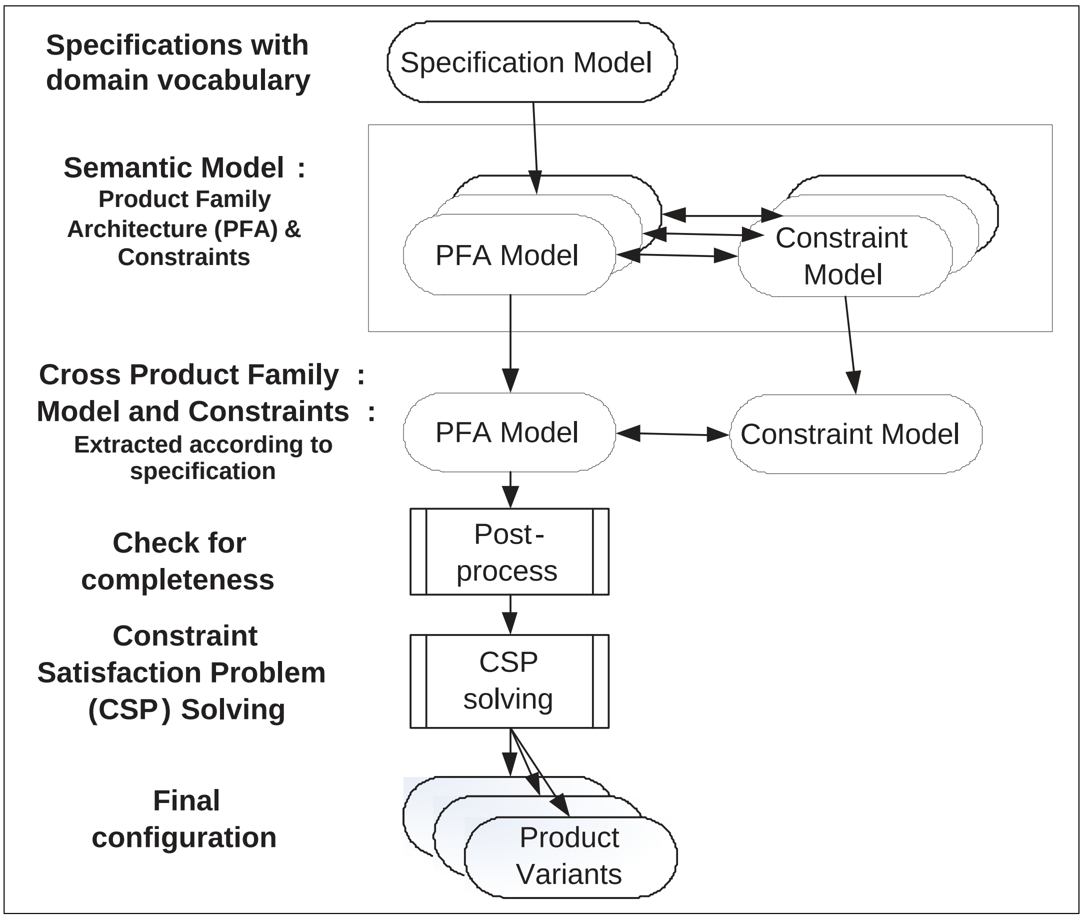

Specifications with domain vocabulary
Specification Model
Semantic Model : Product Family Architecture (PFA) & Constraints
PFA Model
Constraint Model
Cross Product Family : Model and Constraints : Extracted according to specification
PFA Model
Constraint Model
Check for completeness
Post-process
Constraint Satisfaction Problem (CSP) Solving
CSP solving
Final configuration
Product Variants

Figure 3.5: Cross-Product Family-based Configuration

separating the constraints from the PFA. Their dynamic dependency is maintained through the individual modules.

3.4.3 Semantics Extraction

In the cross family product configuration, the requirements specification is partially matched of various product families. In order for the product configuration system to respond to customers rapidly, it is necessary to extract the useful semantic contents embodied in existing product family models, as it is the knowledge used in product variants generation. With semantics modeling, we propose an approach to discover

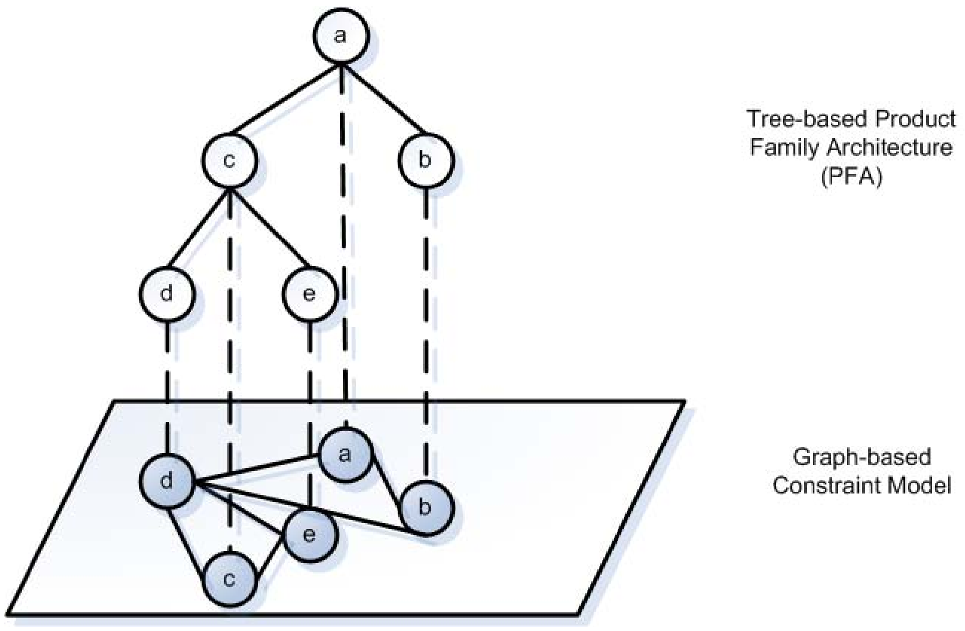

a
c
b
d
e
Tree-based Product Family Architecture (PFA)
a
d
b
e
c
Graph-based Constraint Model

Figure 3.6: Hierarchical Structure of Product Data

and extract semantics from existing product families. The approach consists of three steps:

1. Generate candidate product families by match-making between requirements specification and product family structures;
2. Integrate the useful semantics from the candidate product families;
3. Perform post-processing to ensure the completeness of the semantics.

Candidate Product Family Extraction In our analysis, the requirement specifications are represented as product functions. This corresponds to the functional view; i.e., NF, in our PFA model's module annotation. As the function can usually be decomposed into sub-functions, we organize the requirement specifications into function hierarchical structure. Each function node corresponds to the functional module in the product family tree structure. In the computer configuration example, functional requirement "display" matches the module "monitor". Therefore, we formally define the match-making criterion as:

Definition 5. *A module A of a product family P meets the requirements specification R when the label of a node n in R matches the function of A; i.e., $NF(A) \equiv Label(n)$, where R and P are represented as trees. All the child modules of A are said to match the child nodes of n.*

In product configuration, one functional requirement can be matched by multiple product families, as they may share common modules and the overlapping may occur during the extraction. Therefore, the eligible product families, referred as candidate product families, must be evaluated. Weights are assigned to the requirement specification nodes for the evaluation. The bottom-up weightage assignment is shown in Figure 3.7. Each leaf node is assigned a weight value 1. The weight of the parent node is the sum of the weights of the leaf nodes. Nodes at the same level counted from bottom have the same weight value, which is obtained from the largest weight value of the node in the same level.

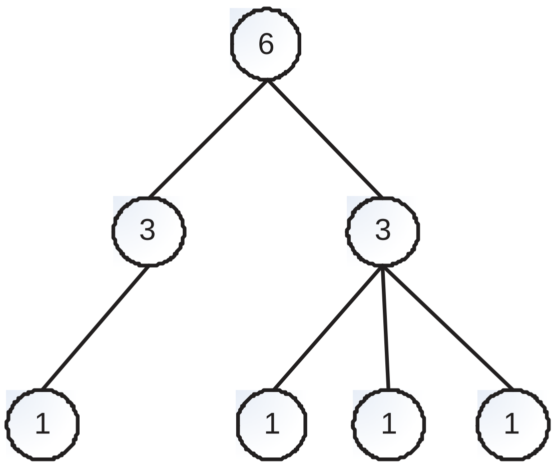


Figure 3.7: Weightage assignment for requirement specification tree

With the above definitions and weight assignment, we propose a search algorithm as shown in Algorithm 1 to discover candidate product families.

Algorithm 1 Candidate Product Families Generation

Input: (1) Requirement specification tree R, with the functional requirements as the tree nodes: N_1, ..., N_m; (2) Product family set PF_1, PF_2, ..., PF_n;

Output: (1) Modules which matches the requirement specification; (2) Candidate product family trees where modules are embodied; (3) Weightage for each candidate product family.

```
1: Searching traverses from root node of $R$: $N_1$
2: Search matches for $N_1$:
3: for $j = 1$ to $n$ do
4:     Search modules that match $N_i$ in $PF_j$
5:     Add $PF_j$ to be candidate product family
6:     Add weightage $W_{N1}$ to $PF_j$
7: end for
8: Stop traversing in $R$ once candidate for $N_1$is found
9: Else, traverse down to $N_2$, one of the child nodes of $N_1$
10: Search matches for $N_2$;
11: Stop traversing to the child nodes of $N_2$ if its corresponding candidate PF is found
12: Traverse to sibling nodes and search for matches
13: Stop when all nodes at the same level are matched
14: Traverse to the children of the nodes that are not matched
15: Continue to traverse in $R$ and search in $PFs$
16: End when no child nodes nor sibling nodes to traverse.
```

Integrating Candidate Product Families From the algorithm, we obtain matchings as shown in Figure 3.8. The nodes with underlined node label are traversed during the search in the search algorithm. The weight of each node is labeled above the nodes. Node $N3$ in this case matches two product families $PF1$ and $PF2$. Therefore, integrating all the candidate product families will result in duplication. Thus, we need to evaluate the candidate product families for node $N3$. In this case, $PF1$ matches $N6$ and $N3$, and $PF2$ matches $N3$ and $N4$. Based on the weight assignment, $PF2$ has an aggregated weightage of 6, which is larger than that of $PF1$. Therefore, $PF2$

is favored for node $N3$. Constraint clusters in $PF2$ will be extracted for $N3$ and $N4$ and clusters in $PF1$ will be extracted for $N6$.

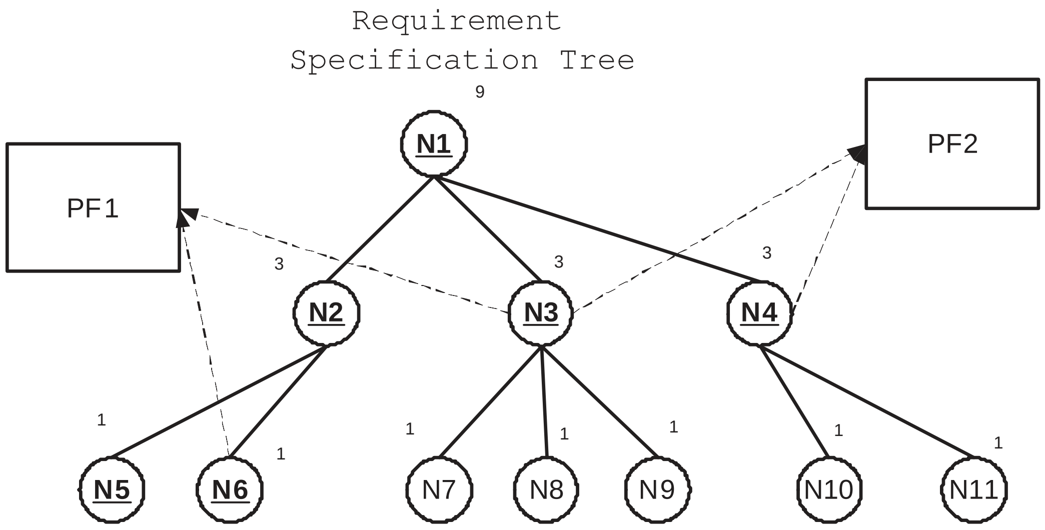


Figure 3.8: Multiple match for one node in a requirement specification tree

In the cross family semantic extraction, the majority of the constraints knowledge can be extracted. However, the constraints extracted are usually not fully completed. This requires knowledge engineers to make minor addition to the constraints generated during product configuration. Manual fine-tuning is necessary.

3.5 Product Life Cycle Cost Model

Product data model provides a fundamental support for product configuration task. However, the model with only design properties is not enough in current manufacturing market as customers are also keen in knowing the estimated cost of the product generated. Therefore, we propose to integrate product lifecycle cost model into our PDM, so as to equip the configurator with cost estimation capability. We adopt the activity-based costing approach [28] to develop a product lifecycle cost model. The lifecycle cost model consists of a set of ABC cost elements, which has its associated activities, cost drivers and resources used. There elements have been systematically defined in [28]. Figure 3.9 shows a model that integrates the product data elements with the lifecycle cost model. Figure 3.10 shows the XML document of an ABC element in the model. Each element has its properties and link with the activity which can be used to derive the cost through cost drivers and resources.

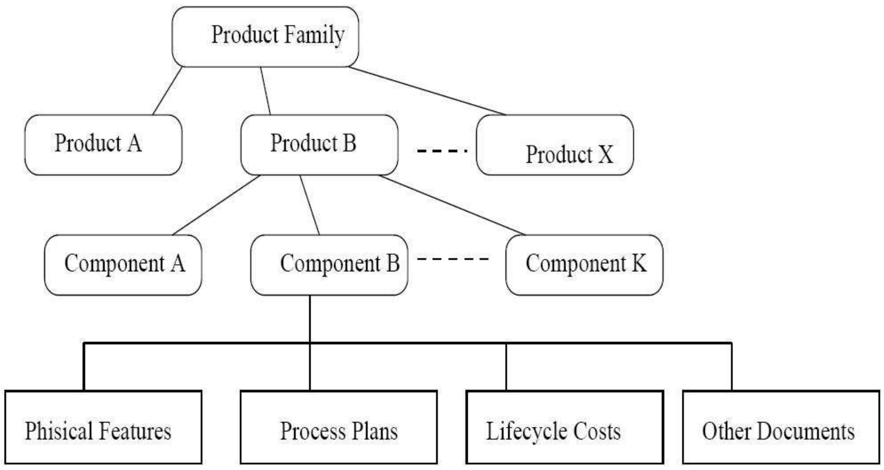
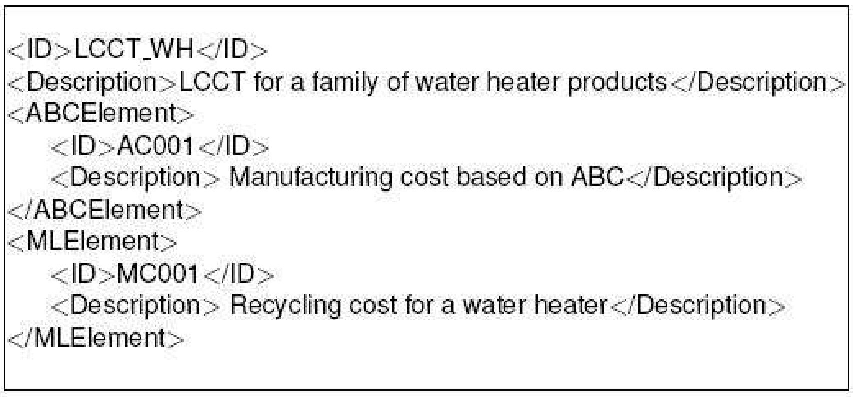


Figure 3.9: Product Data Model with Cost Information



```
<ID>LCCT_WH</ID>
<Description>LCCT for a family of water heater products</Description>
<ABCElement>
    <ID>AC001</ID>
    <Description> Manufacturing cost based on ABC</Description>
</ABCElement>
<MLElement>
    <ID>MC001</ID>
    <Description> Recycling cost for a water heater</Description>
</MLElement>
```

Figure 3.10: XML Document of Cost Data

3.6 Summary

Product family concept has been proved to be an effective way of product organization and product data management for mass customization. Product Family Architecture (PFA) has been defined and the key features have been discussed. Product Data Module (PDM) based on the PFA concept has been developed and shown in the ERD diagram. The entities in the model are described in detail. Computational model is constructed to model the product family *module*, which facilitates the requirement and product features mapping in the configuration task. To address the limitation of the traditional Product Family concept, cross-family model and its corresponding algorithm have been proposed to enable the extraction of features from different product families to meet the customers' unique requirements. Cross-family product data model can significantly increase the possibility of product features combination and enhance the capability of the requirement satisfactions. In addition to the design data, product lifecycle cost data which is critical in the industry design scenario, especially in quotation, has been introduced to integrate into the PDM. This makes the cost estimation of new product configuration possible.

CHAPTER 4 CONSTRAINT MODELING AND ACQUISITION

4.1 Introduction

Product data model has been defined to capture the static information of product for configuration task. In constraint-based product configuration, constraints are the knowledge that drive the configuration process. It represents product data relationships into computational form. Therefore, a constraint model based on the elements of PDM is defined to capture the product design knowledge, as well as the requirement specifications. A generic constraint solver can be used to generate solutions that do not violate the constraints with the available product data. The solutions generated are considered as product variants that meet the requirements. They are configurated according to the design knowledge and industry standards. Thus, in order to properly capture the requirements and design knowledge, the constraint model has to be defined to capture the semantics to provide direct linkage with the product data model and is defined according to the constraint solver syntax. Moreover, as variable ordering and constraint ordering affects the solving efficiency in CSP, the model has to be properly defined to maximize the solving efficiency. On the other hand, we also propose a constraint acquisition method based on the existing product variants, which will be shown and discussed in this chapter.

4.2 Constraint Model

To formalize the product configuration problem as a Constraint-Satisfaction Problem, constraint model is defined to capture the following semantic contents of constraints:

1. Problem variables: i.e., product modules, involved in the constraints, and module parameters must be mapped to the nodes in the Product Data Model;
2. Variety of constraint types: they capture various types of configuration rules;
3. Organization of constraints: i.e., Constraint orderings [35], which is optimized to enhance the CSP solving efficiency.

In product configuration, configuration rules usually need to be presented as N-ary constraints, where more than two problem variables are involved. However, an n-ary constraint can be transformed into several equivalent binary constraints [26]. Therefore, we define a graph-based constraint model using binary constraint as the basic construct. A Constraint cluster is introduced to manage the constraints.

4.2.1 Basic Constraint Construct

Basic constraint construct is the primitive constraint elements. A constraint network is composed of a number of basic constraint constructs. Figure 4.1 shows the three basic constructs of the graph-based constraint diagram. Problem variables are represented as vertices. The edge between the vertices indicates there is a direct constraint relationship between the two connected variables. The weight of an edge shows the number of constraints between the variables. We may interpret the diagram in Figure 4.1 as logical representations:

(a)$\{X \mid_{c_1,\ldots,c_w} \rightarrow Y\}$;

(b)$\{X \mid_{c_1,\ldots,c_{w_1}} \rightarrow Y\}$ & $\{Y \mid_{c_1,\ldots,c_{w_2}} \rightarrow X\}$;

(c)$\{(X,Y) \mid_{c_1,\ldots,c_w}\}$

Note that the C_i $(i \in \{1,\ldots,w\})$ in the representation captures the content of the constraints between the connected variables. This representation also defines the dependent and independent variables in a single constraint, thus, facilitating search space searching in the constraint solving phase.

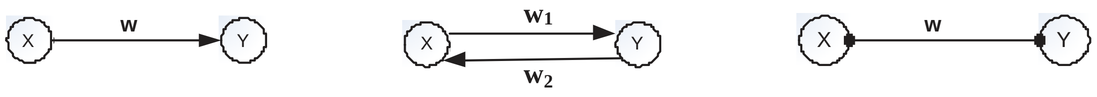


Figure 4.1: Basic Constructs of Constraint Diagram

4.2.2 Constraint Clusters

As configuration problem has strong hierarchical data structure, components in a product structure is not equally important. There are independent and dependent variables. There are critical elements that have strong relationship with other elements. On the other hand, there are some elements that have less links with neighboring elements. As the number of constraints in a product configuration problem is usually very large, proper management of constraint in the constraint graph is necessary to ensure the constraint solving efficiency. Observation shows that constraints relationship among the elements within a product module is denser than between different modules. Therefore, grouping of the elements and their interrelationship within a module would be an intuitive and efficient way of clustering constraints. We introduce the *constraint cluster* and *cluster core* to organize and represent constraints.

Definition 6. *A constraint cluster A in a constraint graph embodies module A or vertex a in the graph, its sub-modules, and the constraints among them; i.e., the graph edges.*

Definition 7. *A cluster core a is the parent module of the vertices in the constraint cluster A.*

From the two definitions, cluster core 'a' together with other cluster cores in the same level and their parent module form another constraint cluster. At a particular cluster, there will be only one cluster core. The size of constraint cluster depends on the complexity of a product. For example, a computer configuration can be considered as a constraint cluster. Its entities in the cluster are monitor, CPU and accessearies. These entities are the cluster cores of their corresponding cluster in their components' assembly level. The forming process continues until there are no more elements for a new cluster. Thus, a set of hierarchical constraint clusters is formed. Figure 4.2 shows the constraint clusters for a constraint graph. Each cluster corresponds to a module in the product family structure. In Figure 4.2(a), there are three constraint clusters B, C, D, that are encapsulated by the dashed contour in the figure. Vertices b, c, d are the cluster cores correspondingly. The constraints between modules are captured by the edges between the cluster cores. Cluster cores b, c, d and vertex a forms cluster A in Figure 4.2(b). Cluster A is defined as the parent cluster of clusters B, C, D. Therefore, the constraint clusters form a tree with vertex a as the root node. The constraint graph is logically represented as:

$$\{cluster\,\overrightarrow{\lambda} : X_\lambda \,|_c\!\rightarrow Y_\lambda\}, \ \overrightarrow{\lambda} = [\lambda_1, \lambda_2, \ldots, \lambda_n]$$

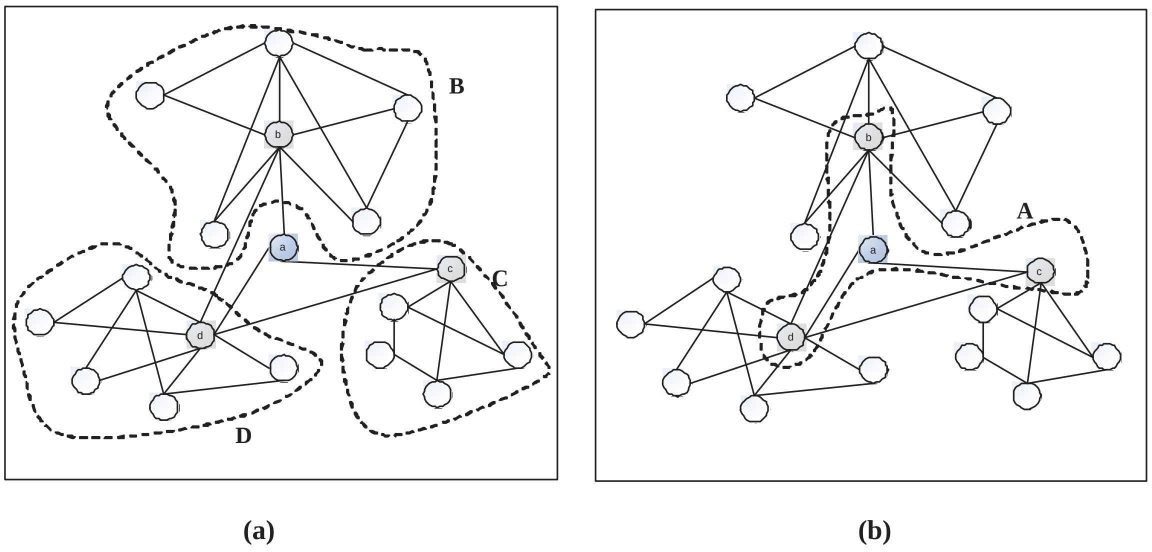


Figure 4.2: Constraint Clusters

In the expression above, λ_i refers to each individual constraint cluster. The significance of the constraint cluster model is two-fold. First, it provides a facility for the exact mapping between the PFA model and the constraints. The cluster cores correspond to the non-leaf nodes in the PFA model. Second, the constraints are ordered with respect to the product family structure. The ordering reflects the constraint distribution in a product family, where constraint clusters and constraint hierarchy are formed. Therefore, it will significantly reduce the size of the search space during the CSP solving and solving efficiency can be significantly enhanced.

4.2.3 Constraint Types

In our constraint model, constraints are divided into two types:

- One type of constraints is the **Compatibility Constraint**. It is also recognized as implication relationship among components. One of the constraint example can be: "IF component A_1 is selected for *module* A and B_1 is selected for *module* B in the configuration, THEN C_1 can be selected for *module* C and C_2 can not be selected." It is usually represented as **IF-Then** rules. It is represented as follows:

$$\text{IF } A = A_1 \text{ AND } B = B_1, \text{THEN } C = C_1 \text{ AND } C \neq C_2$$

- Another type of constraints is the **Equality Constraint** or **Inequality Constraint**. It is mainly to be used to represent the relationship among the module parameters. It is represented in the form of mathematical formula. Parameters are the variables in the formula. An example of equality constraint can be seen in the Motor configuration, where motor power equals to the torque it can provide times the speed. It is represented as $Motor.Power = Torque \times Speed$.

4.2.4 Example of a Configuration Constraint

We use a residential water-heater configuration problem as an example. The product we used is as shown in Figure 4.3. The product structure of the water-heater is presented as in Figure 4.4. It consists of five modules and each has its corresponding attributes. There are three component variants for each module. The attribute values for each component are recorded in Table 4.1. The requirements and design specifications of this water-heater product are described as follow:

- `The shell size must be at least two times as large as the capacity of the tank;`
- `The electric circuit power rating must be equal to power unit output;`
- `The number of turning position of the switch must be equal to Levels of heating rate of the circuit;`

4.2 Constraint Model

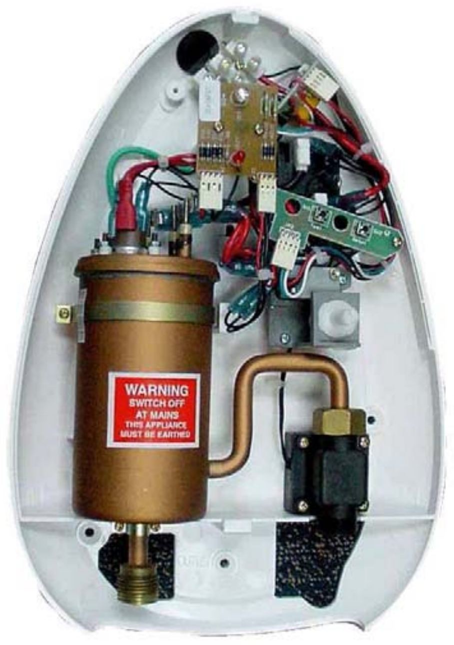


Figure 4.3: A Typical Water Heater Configuration

- `The shell shape must be oval.(The shell is the white plastic casing as shown in Figure 4.4);`
- `Power unit power rating equals to 230V.`

With the product information and the specification, we are able to model the components and attributes into variables and domains, and model the requirements and specifications into constraints. Note that a variable and its corresponding domain will be represented as $Variable\{Domain\}$.

Variables:

- Component level variables: *Tank, PU, Casing, Circuit, Switch*;
- Attribute level variables: *Tcapacity, Tmaterial, PUpowerrating, PUoutput, Ssize, Sshape, Cpowerrating, CLevelNo, Switchno, SwitchType*;

Domain of variables:

4.2 Constraint Model

- Tank {T1, T2, T3}, PU {PU1, PU2, PU3}, Shell {Sh1, Sh2, Sh3}, Circuit {C1, C2, C3}, Switch {Sw1, Sw2, Sw3};
- Tcapacity {1L to 10L}, Tmaterial{Al, Cu, Fe}, PUpowerrating{220V, 230V}, PUoutput{5V,10V,15V, 20V, 30V, 60V},

Constraints:

- `Ssize` $\geq$ `2 x Tcapacity`
- `PUoutput = Cpowerrating`
- `CLevelNo = Switchno`
- `Sshape = Oval`
- `PUpowerrating = 230V`
- `SwitchType = disc'`

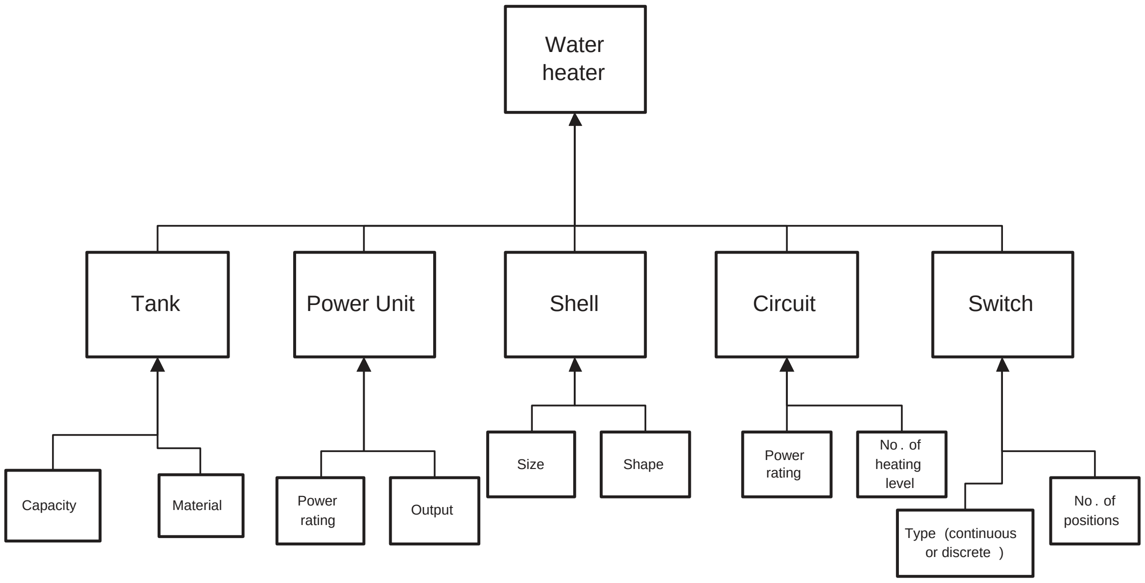


Figure 4.4: Water Heater Product Family Architecture

Table 4.1: Water Heater Data

Module	**Comp**	**Parameter**		**Module**	**Comp**	**Parameter**	
-	-	**Capacity**	**Material**	-	-	**Power**	**Output**
Tank	T_1	6L	Cu	Power	PU_1	220	15V
	T_2	2L	Al		PU_2	220	10V
	T_3	1L	Cu		PU_3	230	10V
-	-	**Input**	**Level**	-	-	**Size**	**Shape**
Circuit	C_1	15V	3	Shell	SS_1	10	Square
	C_2	10V	2		SS_2	15	Round
	C_3	5V	3		SS_3	5	Oval
Switch	SW_1	disc'	2				
	SW_2	cont'	-				
	SW_3	disc'	3				

4.3 Constraint Satisfaction Problem Solving

By modeling the configuration problem as CSP, we can use the generic CSP solving approach to generate solutions. CSP in AI is usually solved via search paradigm. Particularly techniques such as backtracking and constraint propagation are usually used [25]. Backtracking paradigm is an efficient method, where variables are instantiated sequentially, and the validity of the constraint is checked once the variables are instantiated. Backtracking is performed to the most recent instantiated variable when some instantiation does not fit the constraints. Arc Consistency checking as discussed in Section 2.3.2 is implemented to enhance the efficiency and eliminated the thrashing caused by backtracking paradigm. In our configurator, we deploy the open source CSP solver namely Choco [3]. With the configuration constraints defined, the constraint solver is able to generate a list of product variants, which does not violate the constraints. Experiment has been conducted on the water data as shown in Section 4.2.4. The results are shown in Table 4.2. The result shows that the constraint-based configurator is effective and accurate in solving the simulated problem. However, the product data used here is relatively simple. As the product data grows larger and more complex, the solving efficiency may reduce.

Thus, the computational complexity of the solving algorithm is to be investigated to determine the upper bound of the capability of the configurator in future.

Table 4.2: Solutions generated from CSP solver for water heater data

Solution	**Tank**	**Power**	**Circuit**	**Switch**	**Shell**
1	T_1	PU_3	C_2	SW_1	SS_3
2	T_2	PU_3	C_2	SW_1	SS_3

4.4 Constraint Acquisition

As discussed above, configuration knowledge refers to the constraints in the CSP problem. In machine learning study, we are interested to discover patterns from a set of data transactions. Similarly in configuration problem, configuration knowledge is reflected in the product data. Therefore, we can apply appropriate machine learning techniques to discover rules from a set of product data. From a number of product configurations, we should be able to discover the possible relationships among components and parameters by evaluating their probability of the co-existence in the same configuration. The constraint discovery targets in mining the configuration knowledge in a product family. As shown in Figure 4.5, the product variants are instantiated from product family model, where the constraints are imposed. By applying the following proposed approach, the constraints are expected to be partially generated from the variants.

4.4.1 Constraint Acquisition Approach

Association rule mining [51] is a technique to discovery an implication expression of the form $X \rightarrow Y$, where X and Y are disjoint itemsets. It is useful for discovering interesting relationships hidden in data set, which implies an ideal candidate for mining constraint in our case. An association rule takes the following form: Given

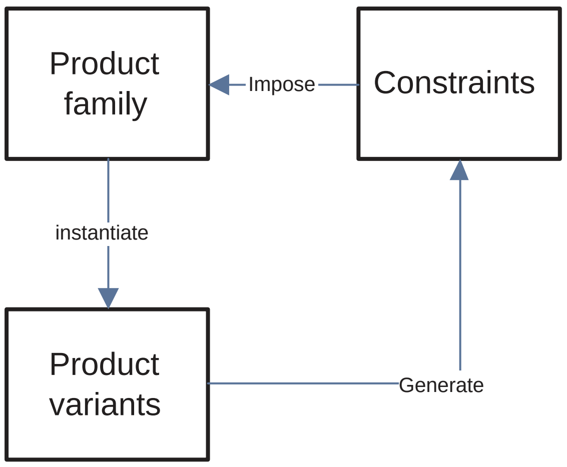


Figure 4.5: Constraint and PF relationships

itemsets: $A\{A1, A2, A3\}$, $B\{B1, B2, B3\}$, $C\{C1, C2, C3\}$, If $A = A1, B = B1$, then $C = C1$ with probability p.

In association rule mining, *support* and *confidence* are commonly used as indicators to assess the probabilities of rules. *Support count* (σ) is defined as the number of datasets for which an association rule covers. For a rule as shown below

$$i_1, i_2, ..., i_n \rightarrow j_1, j_2, ..., j_n$$

Support count of the rule is the number of data sets that contain $\{i_1, i_2, ..., i_n, j_1, j_2, ..., j_n\}$. *Support* ($s$) is usually specified as a percentage of the total number of datasets (T). It is mathematically expressed as

$$s = \frac{\sigma(i_1, i_2, ..., i_n, j_1, j_2, ..., j_n)}{T}$$

Confidence (c) is defined as the number of instances that the association rule predicts correctly. It is expressed as a proportion of all instances to which it applies. It can be represented as

$$c = \frac{\sigma(i_1, i_2, ..., j_1, j_2, ..., j_n)}{\sigma(i_1, i_2, ..., i_n)}$$

In association rule mining, rules with sufficient level of support and confidence that are higher than defined thresholds are considered interested rules.

Common strategy adopted by association rule mining algorithm is to decompose the problem into two major tasks [51]: (i) Frequent Itemset Generation: to find all itemsets with support higher than threshold; (ii) Rule Generation: to extract rules with high confidence from frequent itemset. Algorithms such as `Apriori` have been developed to solve the association analysis problem. The association rule is a probabilistic implication. Therefore, the result obtained will be probabilistic, and verification by the designers is needed after the discovery.

Method description In the product configuration, the compatibility and incompatibility constraints among components are caused by several factors, including the functional compatibility, parameter value relationship, axiomatic design factor [46], etc. The following example shows how parameters' constraint induces components' compatibility constraints.

- `A product consists of A and B: A has parameter 'Length' and B has parameter 'Width'`
- `Variables and domains defined as: A {A1, A2}; B {B1, B2}`
- `A1.Length =10; A2.Length =5; B1.Width=10; B2.Width=5`
- `Constraints: A.Length + B.Width = 15`

From the above problem, we can transverse the attribute constraint into component constraint as:

- Constraints: A1 is compatible with B2, incompatible with B1
- A2 is compatible with B1, incompatible with B2

From the above example, we find that parameter constraints usually induce the component constraints, and it is reflected in the final configuration. Moreover, due to the standardization in manufacturing, discrete attribute value is of more interest than continuous value. E.g. Manufacturers product CPU with speed at several discrete values: 1.6GHz, 1.8GHz, 2.4GHz, instead of a range of continuous speed value between 1.6GHz to 2.4GHz. Other products such as oil field equipments, people use discrete temperature class (K, L, N, S, T, ...) instead of continuous temperature values. Therefore, continuous attribute value is seldom used and automatically acquiring constraints among attribute values will not be of interest in the constraint discovery. This, on the other hand, greatly reduces the computational complexity during rule mining.

Table 4.3: List of transactions for rule mining

Tran	**Tank**	**Power**	**Circuit**	**Switch**	**Shell**
1	T_2	PU_1	C_1	SW_3	SS_2
2	T_1	PU_2	C_2	SW_1	SS_3
3	T_2	PU_3	C_3	SW_3	SS_3
4	T_1	PU_2	C_2	SW_1	SS_2
5	T_3	PU_1	C_1	SW_3	SS_1

In our constraint mining, we consider mining only the compatible and incompatible constraints among components. A complete configuration which consists of a list of composite components is treated as one transaction. Table 4.3 shows a list of transactions for the water-heater configuration.

There are five configurations in Table 4.3. Each item has three possible variants. From the example, we can find out that the item set T_1, PU_2, C_2, SW_1 is a frequent item set, as it has support of 3/5 provided that the threshold is 50%. From the frequent item set, implication rules are further generated. This shows the standard association rule mining technique applied in a small set of configuration data. Experiment will be carried out with larger number of data set to generate the compatibility constraints, i.e. $X \rightarrow Y$.

Moreover, we are also interested in the incompatibility constraint among components, i.e. $X \rightarrow !Y$. Same methodology will be used in generating frequent item set. However, the rule generation step will be different from the above mentioned method. In the compatibility constraint mining, we measure the probability of $p(X \rightarrow Y)$, while in the incompatibility constraint mining, we are looking for $p(X \rightarrow !Y)$, which is equal to $1 - p(X \rightarrow Y)$. Figure 4.6 shows the mining process.

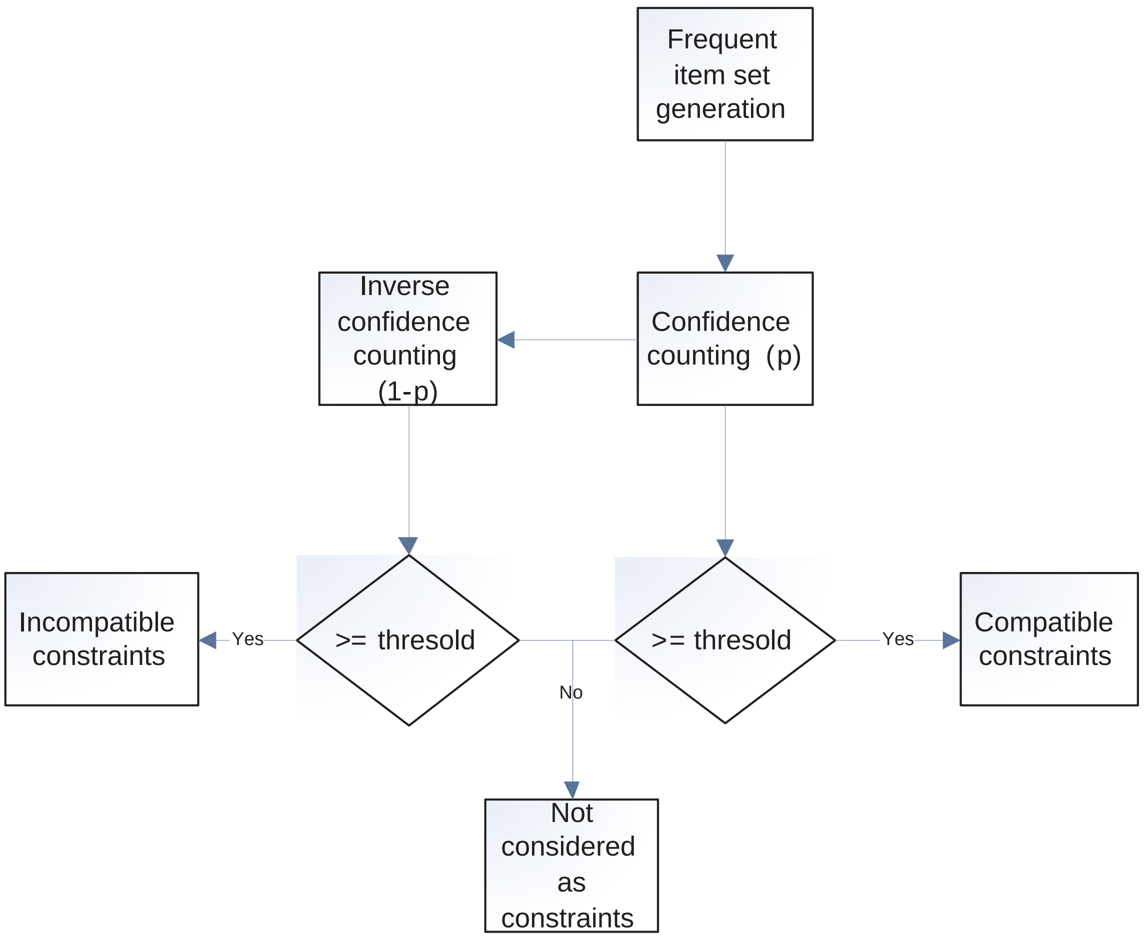


Figure 4.6: Compatibility and incompatibility constraint generation

4.4.2 Experimental Result

In order to test the accuracy of the rules generated, a set of synthetic transactions which consists of 46 pieces of product configurations is generated from the configurator with the following predefined constraints: By applying the method in Section 4.2 upon the data generated and implementing it in the open source data mining software Weka [2], with the following setting:

- Class used: `weka.assocations.Apriori`
- Minimum Confidence: `0.9`
- LowerBoundMinSupport: `0.1 with 0.05 increments.`

'Class' is the predefined association ruling mining library in Weka software. 'Minium Confidence' is to tell the system to generate only rules that has more than 0.9 accuracy on the data. 'LowerBoundMinSupport' with increments defines a range of minimum supports. As we will not know the suitable support value for the given data. By defining a set of supports, we can visually exam the accuracy of the rule generated. In general, those rules with high 'confident' and 'support' values are considered to be more accurate. Top 10 rules are generated in Table 4.4. From the result we see that 60% of the predefined constraints are discovered. It is promising as the experiment only makes use of 46 sets of product configuration. The date size is considered relatively small. In practical product configuration, the number of products under a product family can be hundreds. Moreover, the knowledge generated will be checked and validated by knowledge engineers after knowledge acquisition. Manual fine tuning on the knowledge generated will ensure the accuracy.

Table 4.4: Constraint generated from experiment

NO.	Rule	Confidence	Support
1	Tank = T_3 → Shell = SS_3	90 %	0.457
2	Shell = SS_3 → Tank = T_3	90 %	0.457
3	Tank = T_2 → Shell = SS_1	90 %	0.457
4	Shell = SS_1 → Tank = T_2	90 %	0.457
5	Tank = T_3 Cir = C_3 → Shell = SS_3	90 %	0.196
6	Shell = SS_3 Cir = C_3 → Tank = T_3	80 %	0.196
7	Tank = T_3 Cir = C_1 → Shell = SS_3	90 %	0.196
8	Tank = T_3 Switch = SW_2 → Shell = SS_3	90 %	0.196
9	Shell = SS_3 Cir = C_1 → Tank = T_3	90 %	0.196
10	Cir = C_2 → Switch = SW_2	70 %	0.196

4.5 Summary

In this chapter, we present two main modules in constructing constraint knowledge base. One of the module is constraint model, which is developed to capture configuration knowledge and requirement specifications. The constraint model is presented in the graph format. Basic elements have been developed and shown. To enhance the efficiency in constraint solving in the domain problem, the constraint model is clustered into a hierarchical structure that corresponds to the product structure. Unnecessary backtracking could be eliminated during the constraint solving with the model. A water heater data from local manufacturer has been used to demonstrate the effectiveness of the constraint model. Based on the constraint model and variables from product data model, a generic constraint solver is deployed to solve the configuration problem. Another module applies data mining technique to generate constraints from a set of existing product data. With proper modeling of the transactional data, experiment on Weka has shown that the proposed approach is promising in enabling the automatic knowledge generation.

CHAPTER 5 SYSTEM IMPLEMENTATION

5.1 Introduction

To implement the constraint-based configuration approach, a software system based on our product and constraint model is developed. It is able to capture the requirement inputs and product data. Product knowledge is captured in a knowledge base that can be used by the system during the configuration process. Configuration solving engine is presented to analyze the requirements and generate configurations to meet the requirements. Task coordination system is developed to control the workflow from customer input to the downstream design and manufacturing activities. To enhance the value-add, cost assessment system can be integrated with the product data management system to enable the cost estimation feature. In this chapter, the system architecture of the system will be presented. It consists of the key modules including product data management, constraint management, lifecycle costing, CSP solver, as well as workflow system, each of which will be illustrated in detail.

5.2 System Architecture

Figure 5.1 shows the system architecture that consists of three major system modules. They are Graphic User Interface (GUI) which handles the user-computer interaction, Core Logic which is the core components and its interaction with the modeling and reasoning of the system, and Workflow system which provides coordination of system flow as well as user control. In the system Core Logic module, system starts with the raw customer requirements. Customer requirement analysis module (CRA) is used to interpret the customer requirements into technical requirement specification with well-defined structure. As discussed in Section 2.6 for Figure 2.3, the CRA module is not necessary for configuration system for OEM products as customers usually state the requirements in technical terms. To create and maintain product data in the system, a product data management system (PDM) is developed. As shown in the architecture, the PDM starts with the PF Editor, which is the product family editor. It is used by product engineers to generate product family architecture. It provides a product template to PDM system. Based on the template, product data can be generated by populating data into the template. Moreover, the product family architecture is also the template for the product configuration that will be generated in the later phase. As the product data model is defined to be generic and extensible, lifecycle cost data can be integrated into the product data. The cost data provides data support for the cost assessment module. On the other hand, the system needs to provide features for knowledge engineer to input and fine tune product design knowledge into the system. Therefore, a database is created for storing the constraint-based design knowledge, which is defined and stored according to our constraint model. A constraint editor is developed for knowledge engineers to manually input the design knowledge. Constraint parser then encodes the input into the defined constraint format. With product data, technical requirement specification and the constraint database, a configuration engine is developed to take these data, perform constraint satisfaction solving and generate a set of solutions that meets the requirements. Iterations are performed through cost assessment module to select the solutions with the optimal cost.

In the system core logic module, key components including PDM, Configuration engine, constraint editor and lifecycle cost estimation has been developed and will be presented in detail in the following sections. As the configuration system involves different users and has to follow data flow sequence, a workflow system is defined to coordinate the system flow and control user access. In our system, we develop a generic workflow system which allows users to define their own activities, sequence and user access. The system is presented in detail in the following section.

5.3 Product Data Management System

As discussed in 5.2, Product Data Management system allows users to create product family architecture and populate product data based on the architecture. This section will discuss in detail the components in the PDM. Screen shots will be shown to demonstrate the GUI. UML representation of the programming classes will be shown to reflect the internal programming structrue.

5.3.1 PDM System Architecture

Figure 5.2 shows the system architecture of the PDM application. Presentation layer is the GUI that provides the interaction between users and the system. The core module is the ‘business logic layer’. PDM core logic is the key component that includes data management functions such as ‘file I/O’ and ‘configuration file parser’ which provides interface with the external documents. It also includes functions for interaction with the database. PFA creator is used to interact with the database controller to store the product family architecture input from the GUI. The data is

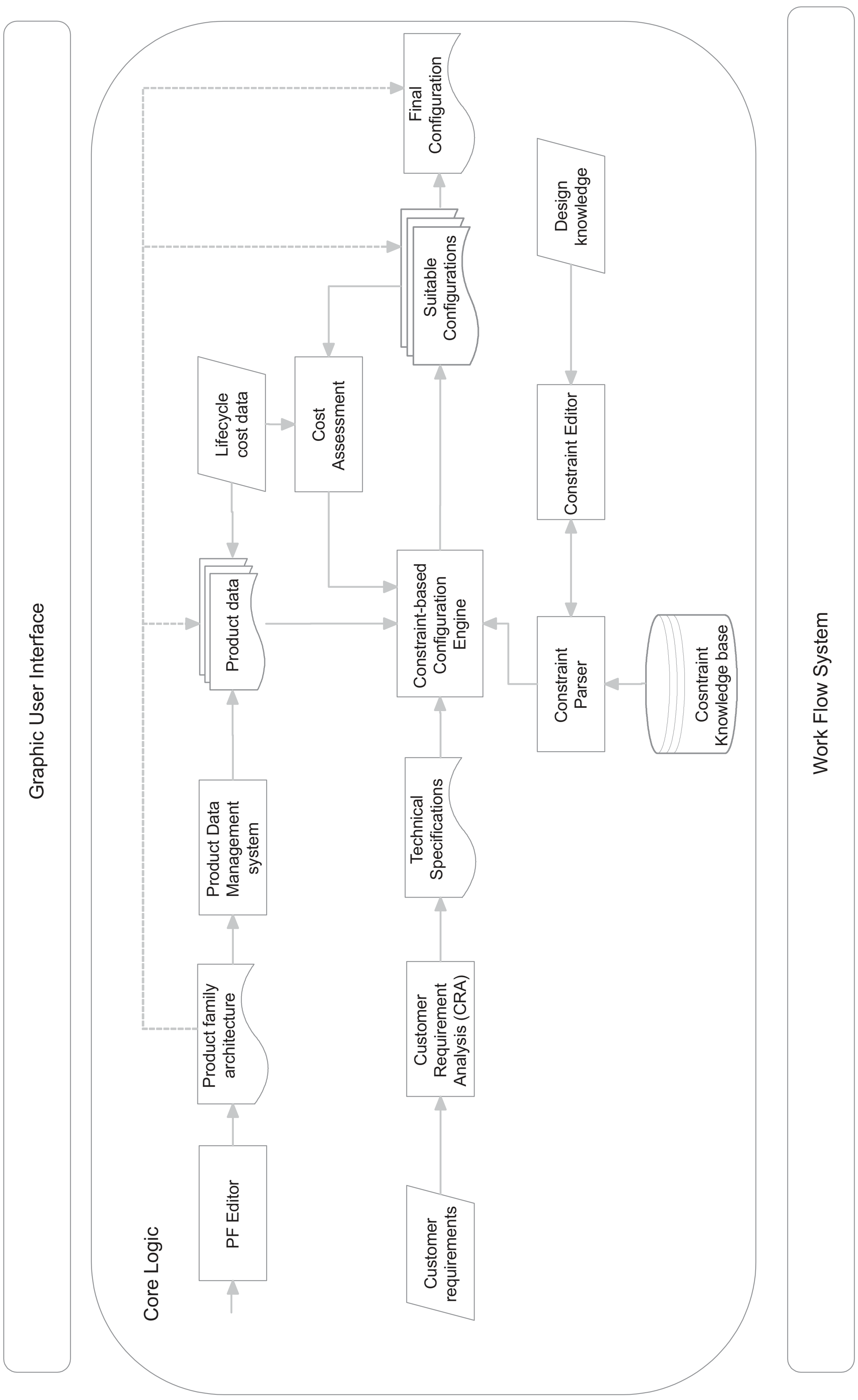


Figure 5.1: Product Configuration System Architecture

hierarchical structure where lower level modules are encapsulated in the parent level module. Data generator handles the product instance data with database controller. It populates the product data based on the product family architecture and transfer the formatted data to the database controller. 'Query handler' is developed to handle the product or product family query from GUI. Product data is retrieved from database through this handler. 'Document organizer' is used for associating the product related documents including 3D drawing to the product data. Documents can be uploaded and retrieved through this module. 'Data access layer' provides interface for linking MS SQL Database and the 'business logic layer'. In this layer, MS SQL database controller is used to parse and transfer action commands from 'business logic layer' to the database API. Database wrapper API is used to interface with the database, so that command input can be simplified and more robust.

5.3.2 Class Diagram of PDM System

The key component of the PDM system development is the application-database communication. Figure 5.3 shows the UML diagram of the product data, which is stored in the database as table format. All the classes as shown implement the `ICloneable` interface to allow data duplication and reuse. The class diagram fully implements the hierarchical structure of the product data model defined in Chapter 3.

In Data Access Layer, MS SQL database controller is used to provide an interface to perform operation on each individual data table that is stored in MS SQL database. This module contains the classes as shown in Figure 5.4, which has been implemented in the system. The base class `SqlWrapperBase` implements the existing interface `ISqlWrapperBase`, which is an interface in Database wrapper that interact with MSSQL database. The child classes as shown contains methods for performing database query, insertion and retrieval for each individual table.

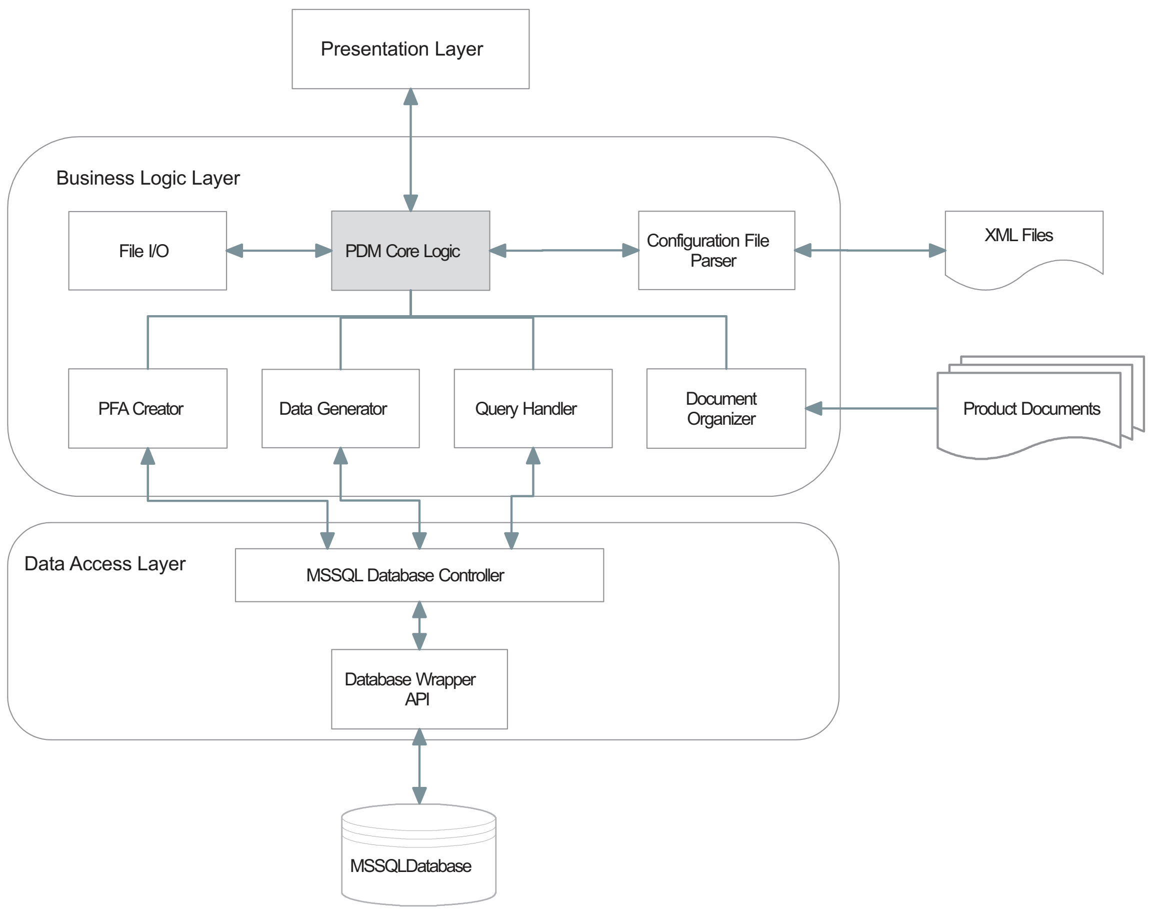


Figure 5.2: PDM System Architecture

5.3.3 PDM System Walk-through

Overview By implementing the system architecture and the class diagrams, a software prototype has been developed. The system overview is shown in Figure 5.5. User can start with either creating a new product family architecture or loading one from existing product document or database. Once activated, a product family editor window will show up. It allows user to edit product modules and module parameters, and define parameter value range, etc. Product family architecture information can be saved to a document or database when user finishes editing. Next step, in order to create new product data, user can enter into 'product editor' window by selecting the existing product family architecture as template. Alternatively, user is allowed

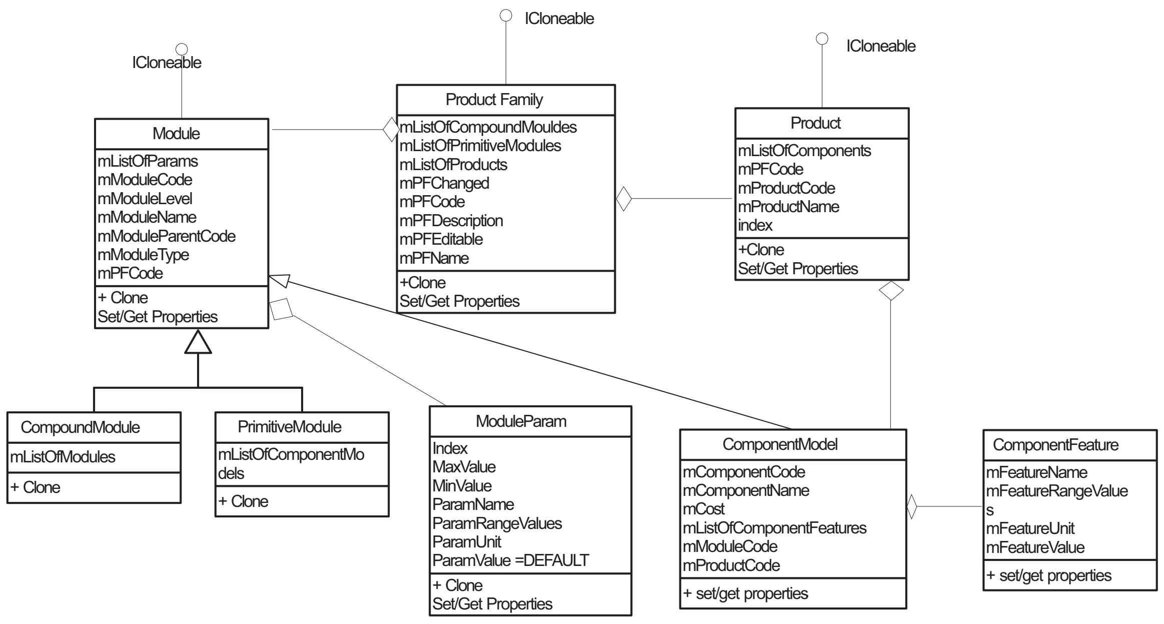


Figure 5.3: Class Diagram of Product Data Model

to load existing product data to view or edit. In the 'product editor', user can edit module information, populate data for module and module parameters. Moreover, user can upload product drawings, documents to associate with the product data. This is usually for product designer where 3D product drawing is necessary. A 3D viewer is developed and integrated into the system to allow users to view common 3D drawing, including SolidWorks, AutoCad, ProE, etc. All the product data can be saved into product documents or database. The GUI for the software components will be shown in the following section.

PF Editor Figure 5.6 shows the GUI of product family editor. A tree structure that displays the product family skeleton is shown in the left panel. User can perform regular module operation by right clicking to activate the context panel. Product family's detail module and parameter information is shown in the right panel. From the right panel, modules and parameters are represented in a table structure. Attributes such as module type, module code and so on can be manually keyed in and

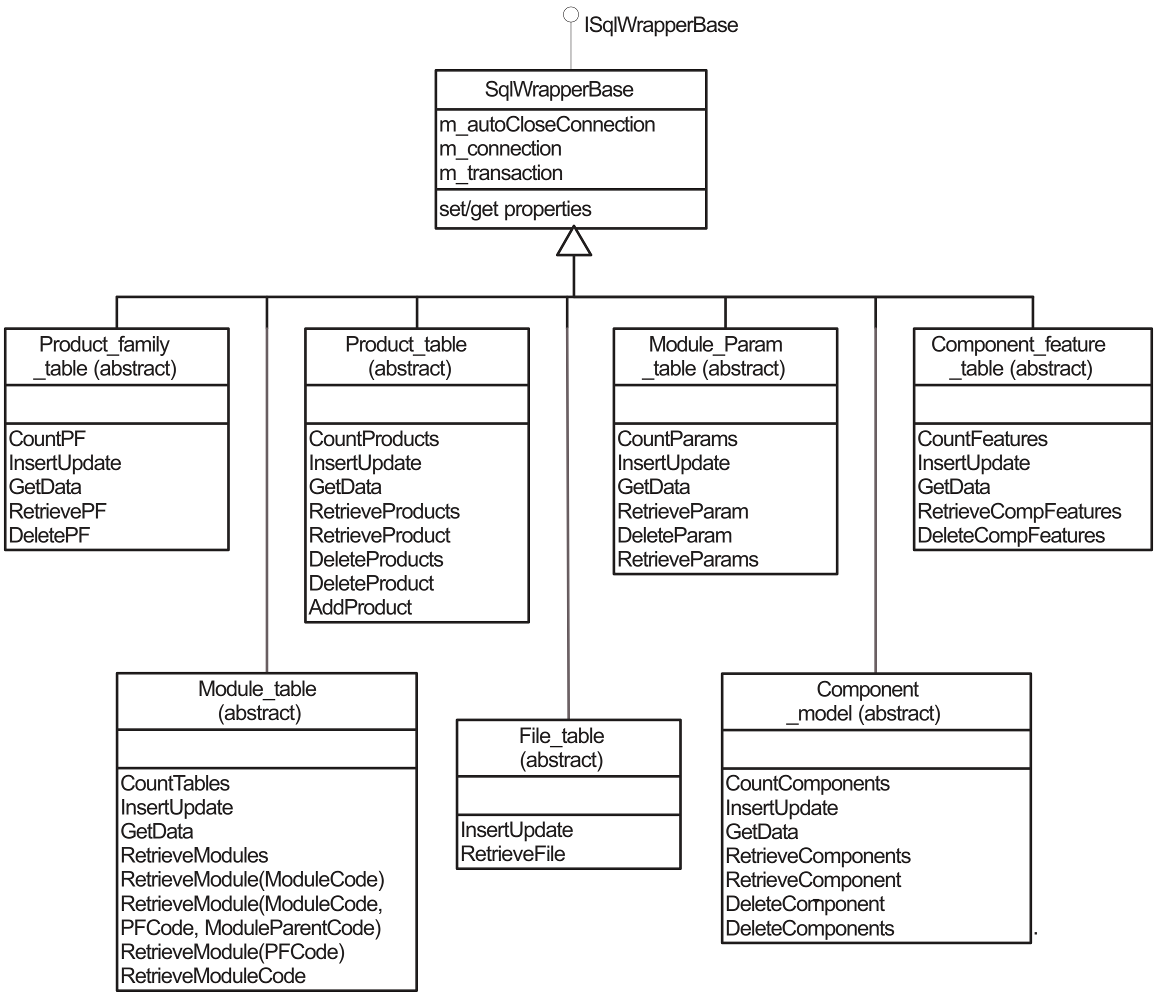


Figure 5.4: MSSQL Controller Class Diagram

save. In the menu bar, shortcuts for the operations is created. User can open or load product family from database by using the combo box as shown. 'PF editor' allows user to view and upload product related documents to the system. Figure 5.7 shows the 3D drawing display for the modules. The 3D viewer provides simple manipulation tool for user to rotate and slide the drawings to enhance visibility.

Product Data Editor Figure 5.8 shows the starting dialog while creating a new product. User is prompted to select a product family architecture as product template. Product Data Editor is shown in Figure 5.9. It is similar to product

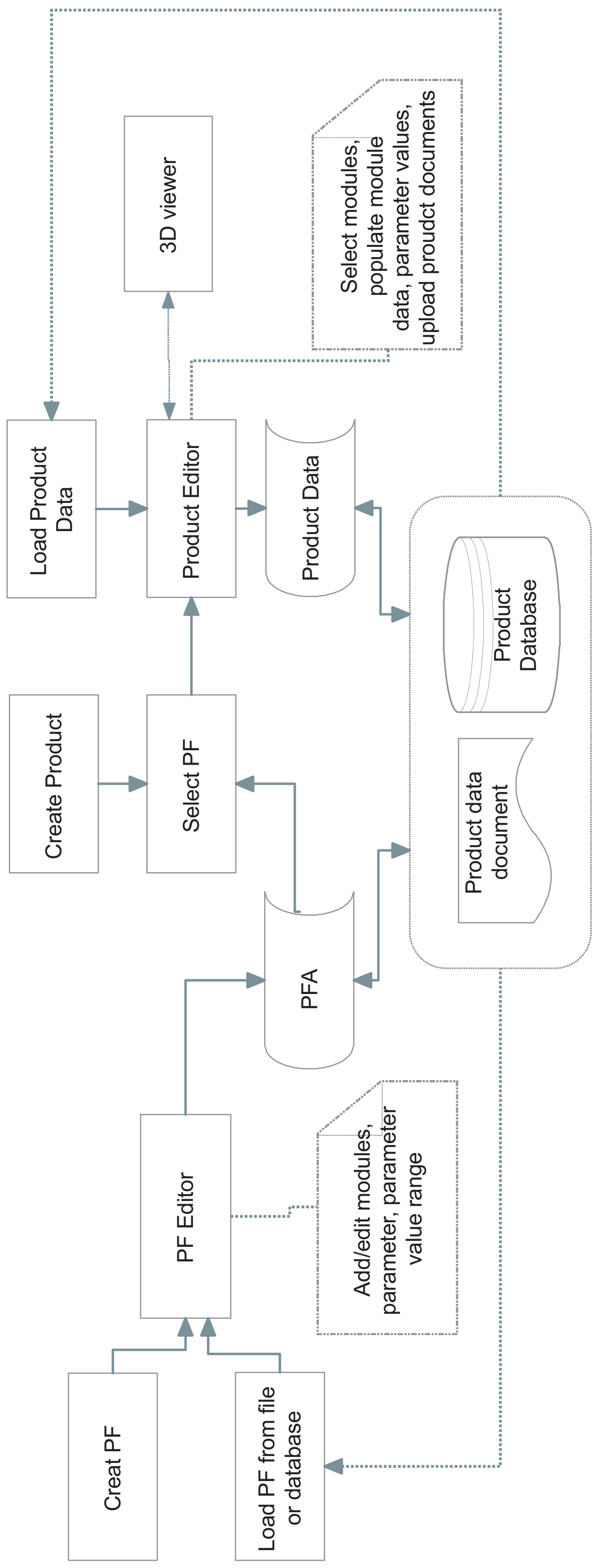
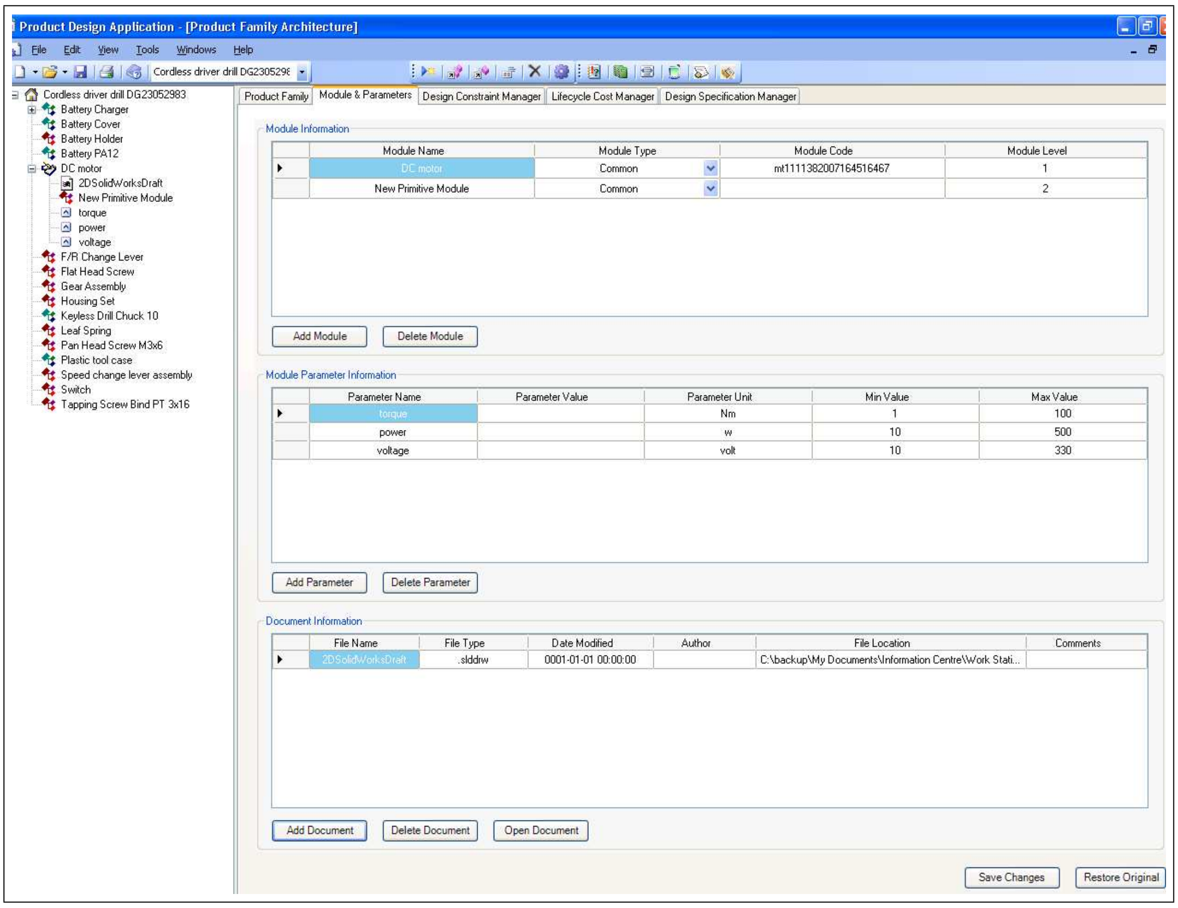

Creat PF
Load PF from file or database
PF Editor
Add/edit modules, parameter, parameter value range
PFA
Create Product
Select PF
Product data document
Product Database
Load Product Data
Product Editor
Product Data
3D viewer
Select modules, populate module data, parameter values, upload proudct documents

Figure 5.5: Product Configuration System Architecture

5.3 Product Data Management System


Product Design Application - [Product Family Architecture]
File Edit View Tools Windows Help
Cordless driver drill DG2305298
Product Family
Module & Parameters
Design Constraint Manager
Lifecycle Cost Manager
Design Specification Manager
Add Module
Delete Module
Add Parameter
Delete Parameter
Add Document
Delete Document
Open Document
Save Changes
Restore Original

- Cordless driver drill DG23052983
  - Battery Charger
  - Battery Cover
  - Battery Holder
  - Battery PA12
  - DC motor
    - 2DSolidWorksDraft
    - New Primitive Module
    - torque
    - power
    - voltage
  - F/R Change Lever
  - Flat Head Screw
  - Gear Assembly
  - Housing Set
  - Keyless Drill Chuck 10
  - Leaf Spring
  - Pan Head Screw M3x6
  - Plastic tool case
  - Speed change lever assembly
  - Switch
  - Tapping Screw Bind PT 3x16

Module Information

Module Name	Module Type	Module Code	Module Level
DC motor	Common	mt1111382007164516467	1
New Primitive Module	Common		2

Module Parameter Information

Parameter Name	Parameter Value	Parameter Unit	Min Value	Max Value
torque		Nm	1	100
power		w	10	500
voltage		volt	10	330

Document Information

File Name	File Type	Date Modified	Author	File Location	Comments
2DSolidWorksDraft	.slddrw	0001-01-01 00:00:00		C:\backup\My Documents\Information Centre\Work Stati...	

Figure 5.6: Product Family Editor GUI

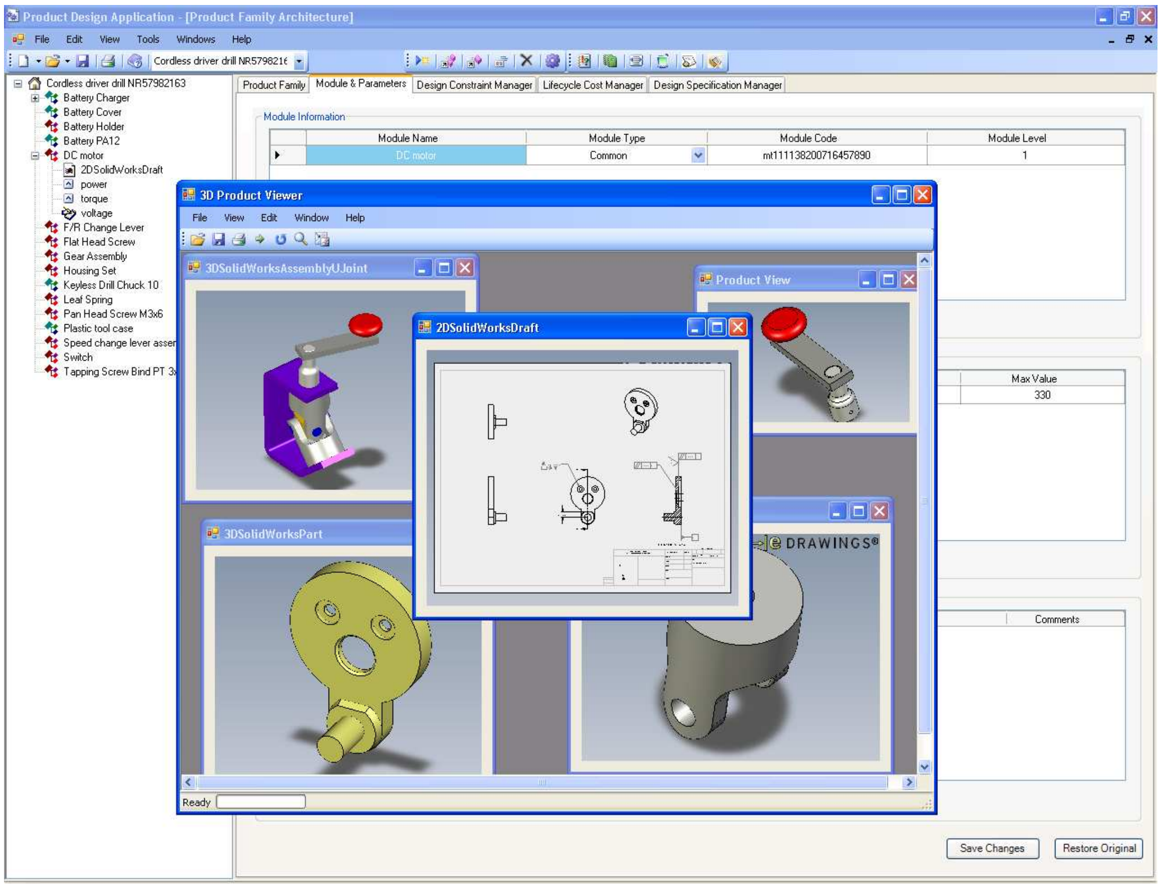
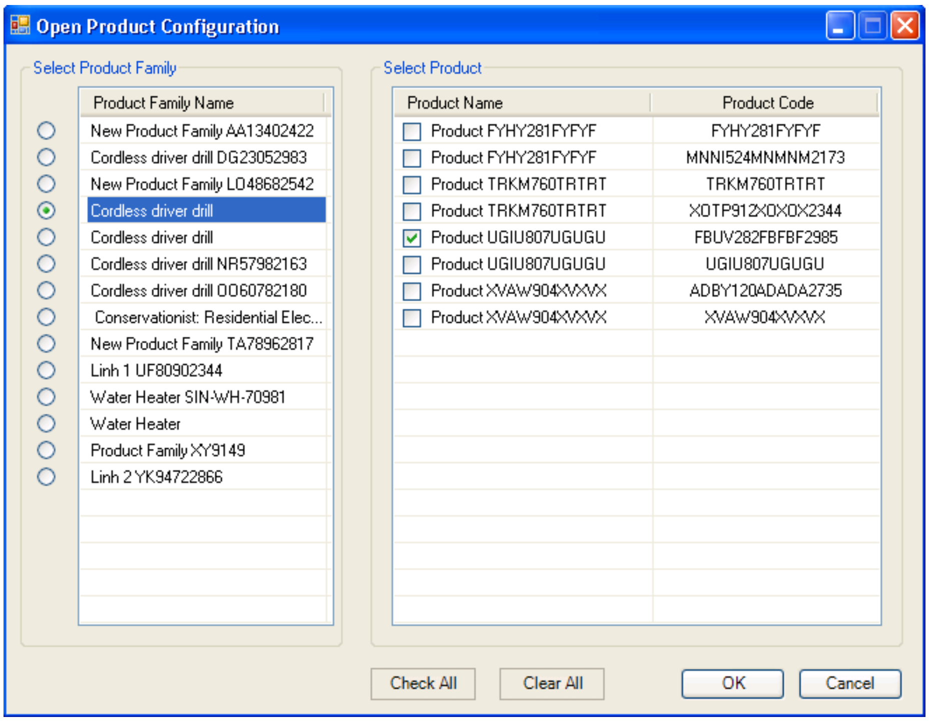


Figure 5.7: 3D CAD Drawing Display

family editor. Product hierarchy is presented in the left panel. Detail component information can be viewed or edited in the right panel.

5.4 Constraint Manager

Another key component in product configuration system is the constraint management module. It consists of the constraint editor and constraint database. With the theoretical support discussed in Chapter 4, constraint model can be developed into the ERD diagram as shown in Figure 5.10. In the figure, constraint consists of `rule` and `equation`. Both of the two types of constraints contain elements called `expression`. The `expression` is a general entity which covers various kinds of


Open Product Configuration

Select Product Family

	Product Family Name
○	New Product Family AA13402422
○	Cordless driver drill DG23052983
○	New Product Family LO48682542
●	Cordless driver drill
○	Cordless driver drill
○	Cordless driver drill NR57982163
○	Cordless driver drill OO60782180
○	Conservationist: Residential Elec...
○	New Product Family TA78962817
○	Linh 1 UF80902344
○	Water Heater SIN-WH-70981
○	Water Heater
○	Product Family XY9149
○	Linh 2 YK94722866

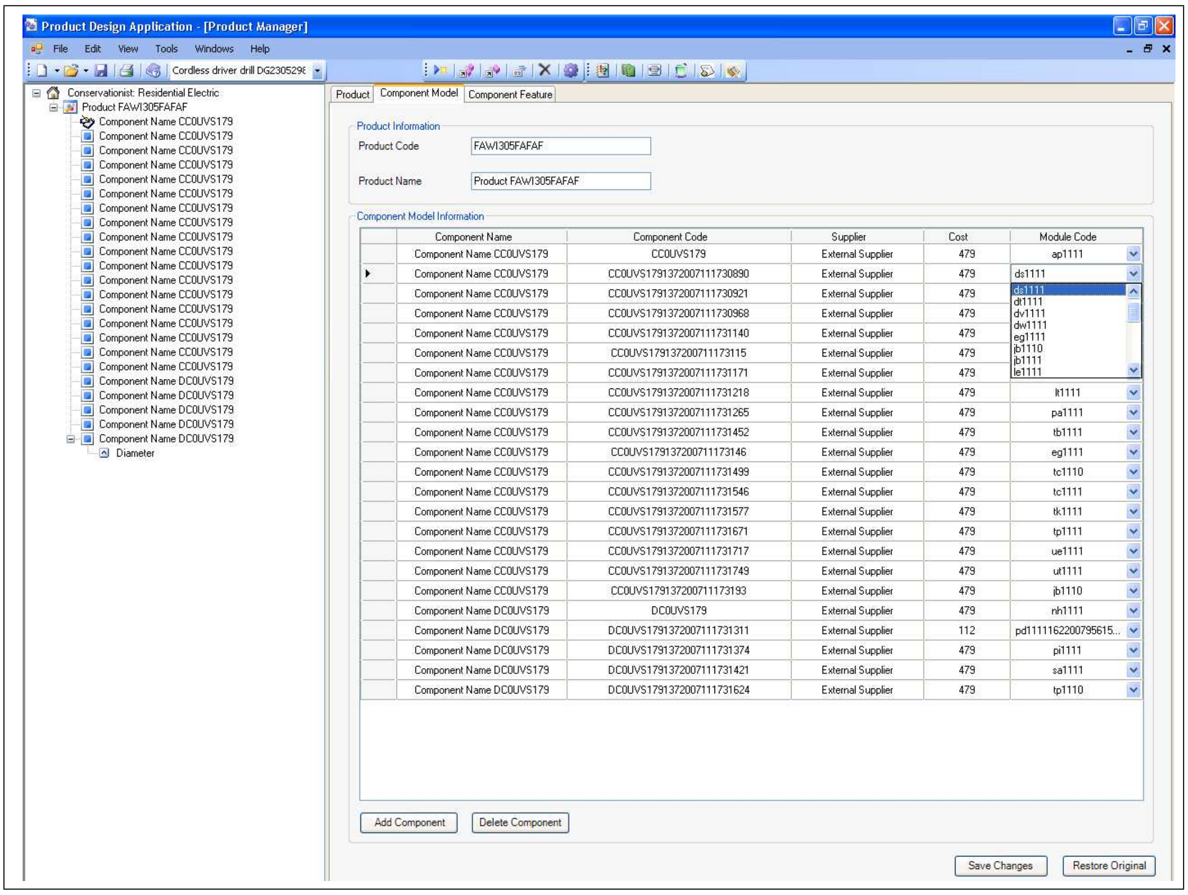
Select Product

	Product Name	Product Code
☐	Product FYHY281FYFYF	FYHY281FYFYF
☐	Product FYHY281FYFYF	MNNI524MNMNM2173
☐	Product TRKM760TRTRT	TRKM760TRTRT
☐	Product TRKM760TRTRT	XOTP912XOXOX2344
✓	Product UGIU807UGUGU	FBUV282FBFBF2985
☐	Product UGIU807UGUGU	UGIU807UGUGU
☐	Product XVAW904XVXVX	ADBY120ADADA2735
☐	Product XVAW904XVXVX	XVAW904XVXVX

Figure 5.8: Starting Dialog for Open a Product Configuration

constraint representation. For example, "`Power` = `Torque` × `Speed`" is a binary expression, which consists of expression "`Power`" and expression "`Torque` × `Speed`", with an `operator` "=". With the constraint model properly defined with the database, a constraint editor GUI has been developed and the GUI is shown in Figure 5.11. In the top portion of the dialog, buttons are used for the operations for rules and operators of equations are also listed in the interface. User can select the existing modules to add to the constraint by selecting the combo box. Constraints are updated and shown in the text box in the lower portion. They can be saved to database and XML documents, which will be used for configuration in the later phase.

5.4 Constraint Manager



Conservationist: Residential Electric
Product FAWI305FAFAF
Component Name CC0UVS179
Component Name CC0UVS179
Component Name CC0UVS179
Component Name CC0UVS179
Component Name CC0UVS179
Component Name CC0UVS179
Component Name CC0UVS179
Component Name CC0UVS179
Component Name CC0UVS179
Component Name CC0UVS179
Component Name CC0UVS179
Component Name CC0UVS179
Component Name CC0UVS179
Component Name CC0UVS179
Component Name CC0UVS179
Component Name CC0UVS179
Component Name CC0UVS179
Component Name CC0UVS179
Component Name DC0UVS179
Component Name DC0UVS179
Component Name DC0UVS179
Component Name DC0UVS179
Component Name DC0UVS179
Diameter

Product Information

Product Code	FAWI305FAFAF
Product Name	Product FAWI305FAFAF

Component Model Information

	Component Name	Component Code	Supplier	Cost	Module Code
	Component Name CC0UVS179	CC0UVS179	External Supplier	479	ap1111
▶	Component Name CC0UVS179	CC0UVS1791372007111730890	External Supplier	479	ds1111
	Component Name CC0UVS179	CC0UVS1791372007111730921	External Supplier	479	
	Component Name CC0UVS179	CC0UVS1791372007111730968	External Supplier	479	
	Component Name CC0UVS179	CC0UVS1791372007111731140	External Supplier	479	
	Component Name CC0UVS179	CC0UVS179137200711173115	External Supplier	479	
	Component Name CC0UVS179	CC0UVS1791372007111731171	External Supplier	479	
	Component Name CC0UVS179	CC0UVS1791372007111731218	External Supplier	479	lt1111
	Component Name CC0UVS179	CC0UVS1791372007111731265	External Supplier	479	pa1111
	Component Name CC0UVS179	CC0UVS1791372007111731452	External Supplier	479	tb1111
	Component Name CC0UVS179	CC0UVS179137200711173146	External Supplier	479	eg1111
	Component Name CC0UVS179	CC0UVS1791372007111731499	External Supplier	479	tc1110
	Component Name CC0UVS179	CC0UVS1791372007111731546	External Supplier	479	tc1111
	Component Name CC0UVS179	CC0UVS1791372007111731577	External Supplier	479	tk1111
	Component Name CC0UVS179	CC0UVS1791372007111731671	External Supplier	479	tp1111
	Component Name CC0UVS179	CC0UVS1791372007111731717	External Supplier	479	ue1111
	Component Name CC0UVS179	CC0UVS1791372007111731749	External Supplier	479	ut1111
	Component Name CC0UVS179	CC0UVS179137200711173193	External Supplier	479	jb1110
	Component Name DC0UVS179	DC0UVS179	External Supplier	479	nh1111
	Component Name DC0UVS179	DC0UVS1791372007111731311	External Supplier	112	pd1111162200795615...
	Component Name DC0UVS179	DC0UVS1791372007111731374	External Supplier	479	pi1111
	Component Name DC0UVS179	DC0UVS1791372007111731421	External Supplier	479	sa1111
	Component Name DC0UVS179	DC0UVS1791372007111731624	External Supplier	479	tp1110

ds1111
dt1111
dv1111
dw1111
eg1111
jb1110
jb1111
le1111

Figure 5.9: Product Data Editor GUI

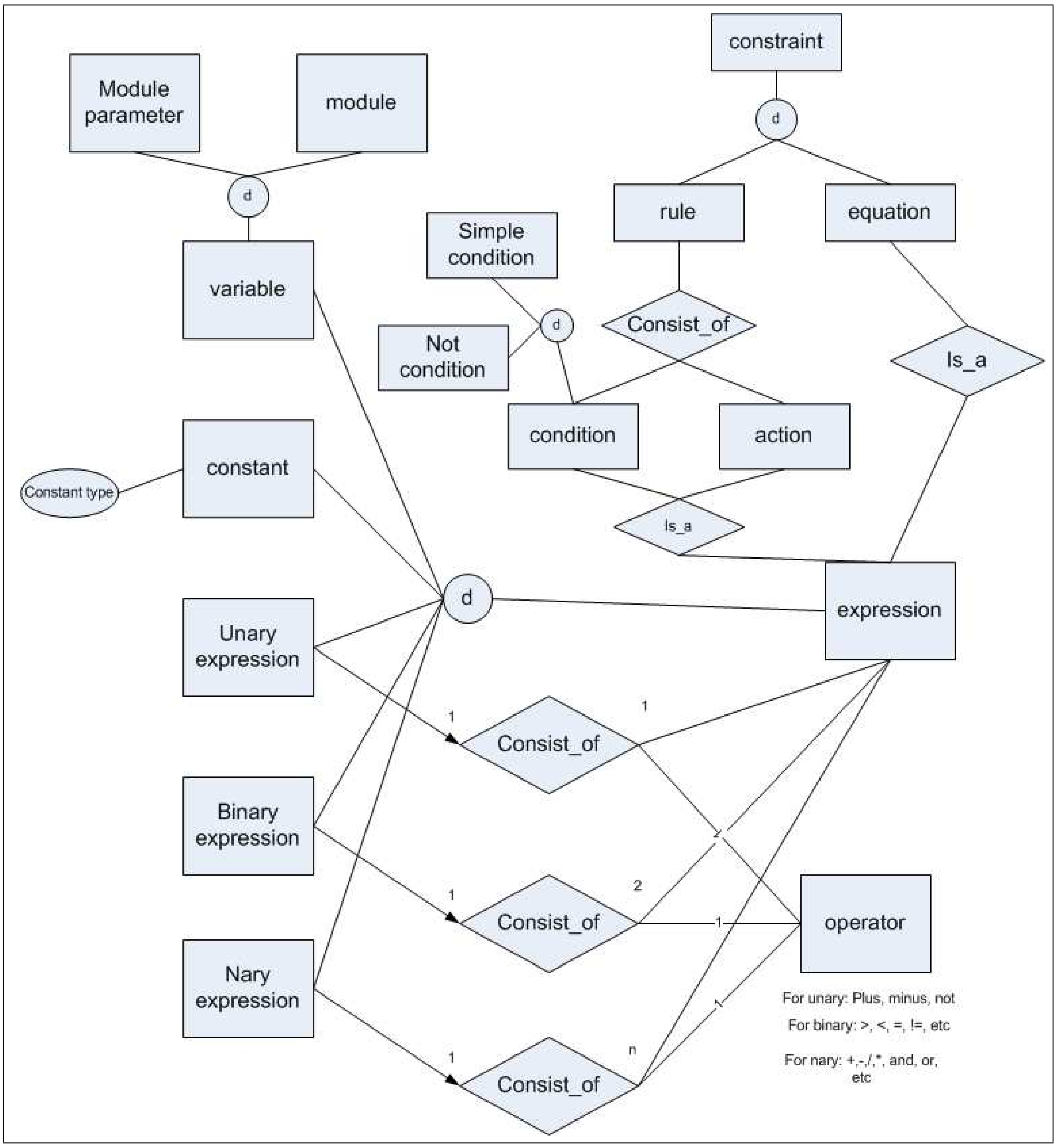
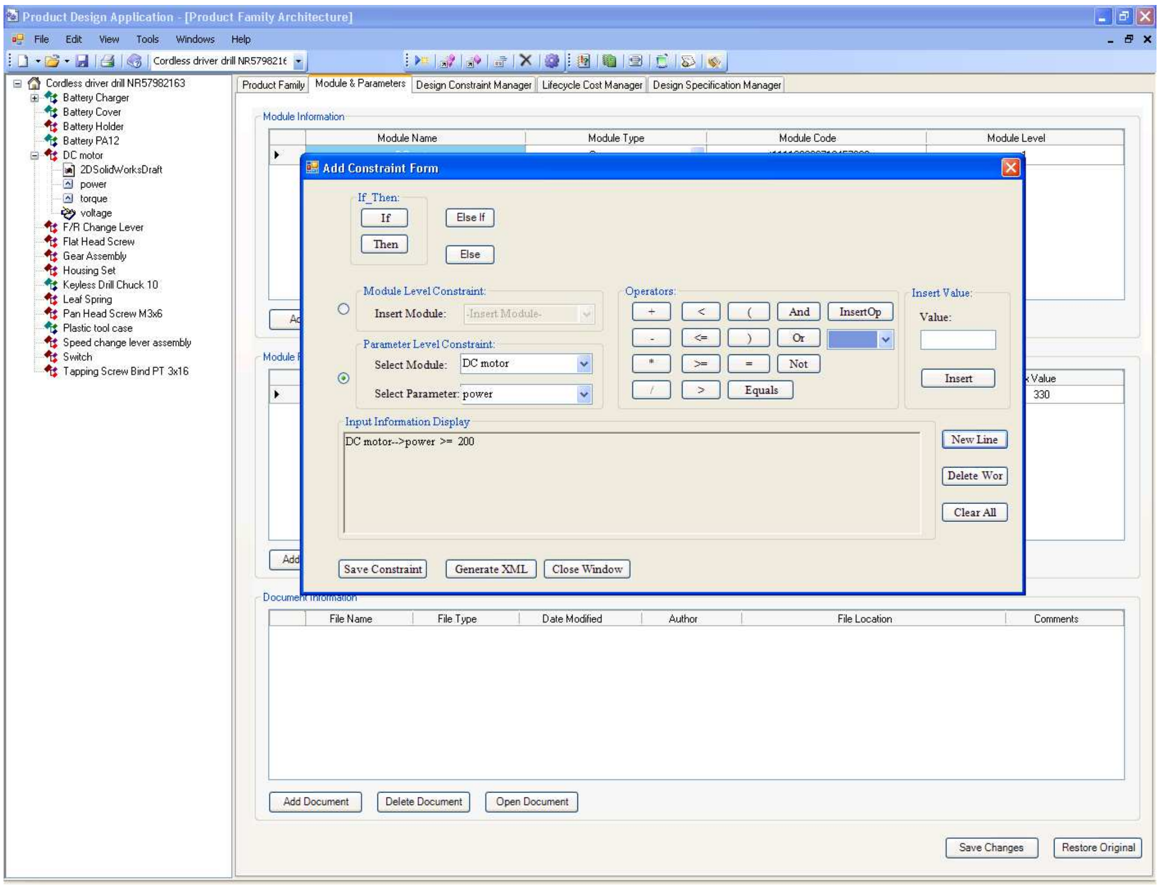


Figure 5.10: Constraint Data Model



Cordless driver drill NR57982163
- Battery Charger
- Battery Cover
- Battery Holder
- Battery PA12
- DC motor
  - 2DSolidWorksDraft
  - power
  - torque
  - voltage
- F/R Change Lever
- Flat Head Screw
- Gear Assembly
- Housing Set
- Keyless Drill Chuck 10
- Leaf Spring
- Pan Head Screw M3x6
- Plastic tool case
- Speed change lever assembly
- Switch
- Tapping Screw Bind PT 3x16

Insert Module: -Insert Module-

Select Module: DC motor

Select Parameter: power

```
DC motor-->power >= 200
```

Value: 330

Figure 5.11: Constraint Editor GUI

5.5 Product Configuration Engine

PDM system and constraint manager is defined to interface with configuration engine as shown in the system architecture in Figure 5.1. As discussed, configuration engine is based on the CSP problem solving approach. Thus, to interface with PDM and constraint manager, variables and constraints are defined according to the CSP syntax. Figure 5.12 shows the steps in programming the configuration engine. A configuration problem is defined in the programming environment. Variables are defined and included in the problem. Different types of variables including integer, continuous value, set and enumeration can be defined. Next, variable domain has to be defined based on the type of variable. Constraint is then defined. It will also need to be included into the problem. Function `post` is used to post a defined constraint into the problem. With all necessary constraints and variables defined, function `Problem.SolveAll()` in CSP solver can be called to generate possible solutions.

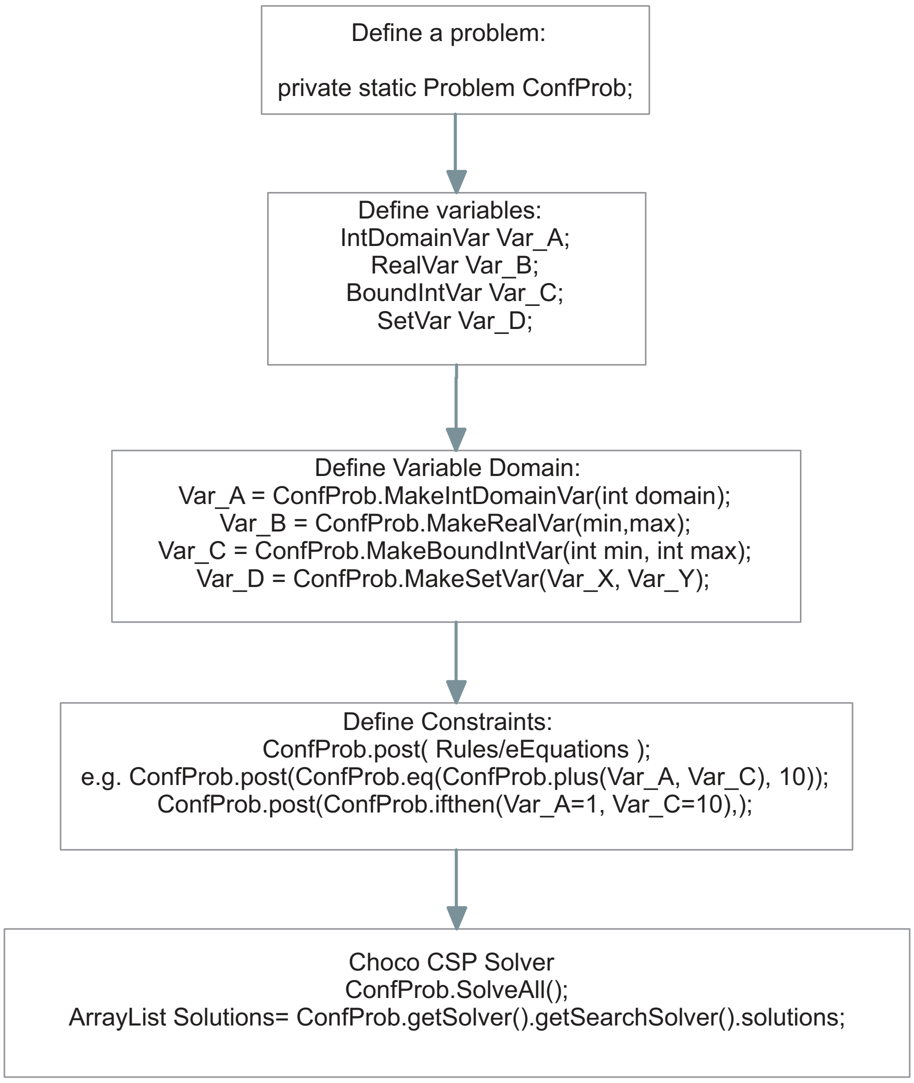

Define a problem:
private static Problem ConfProb;
Define variables:
IntDomainVar Var_A;
RealVar Var_B;
BoundIntVar Var_C;
SetVar Var_D;
Define Variable Domain:
Var_A = ConfProb.MakeIntDomainVar(int domain);
Var_B = ConfProb.MakeRealVar(min,max);
Var_C = ConfProb.MakeBoundIntVar(int min, int max);
Var_D = ConfProb.MakeSetVar(Var_X, Var_Y);
Define Constraints:
ConfProb.post(Rules/eEquations);
e.g. ConfProb.post(ConfProb.eq(ConfProb.plus(Var_A, Var_C), 10));
ConfProb.post(ConfProb.ifthen(Var_A=1, Var_C=10),);
Choco CSP Solver
ConfProb.SolveAll();
ArrayList Solutions= ConfProb.getSolver().getSearchSolver().solutions;

Figure 5.12: Configuration Engine Programming Flowchart

Configuration Engine UML In this implementation, we focus on modeling the configuration problem into CSP solving paradigm. Conventional CSP solving approaches including backtracking and consistency check are adopted in the system. Thus, we deploy the Java-based CSP solving library `choco` to be the fundamental solver. The class diagram in Figure 5.13 shows the major classes in modeling the configuration problem with the library. `ConfigEng` is our application class that consists of the main function. In this class, the four-step modeling as discussed earlier is coded. To define a configuration problem, the `problem` class is instantiated. This class has the fields including variables and constraints within this problem. Methods are called out to include variables into the problem. Different types of variables are separated by using three different methods. They are `makeIntVar()`, which adds integer domain variables; `makeRealVar()`, which adds continuous domain variables; `makeSetVar()`, which add variables, of which value is a set. Similarly, for function `post()`, which add constraints to the problem, similar steps are taken. Interfaces for `Constraint` and `Var` are defined to enable the tracking among problem, constraint and variable.

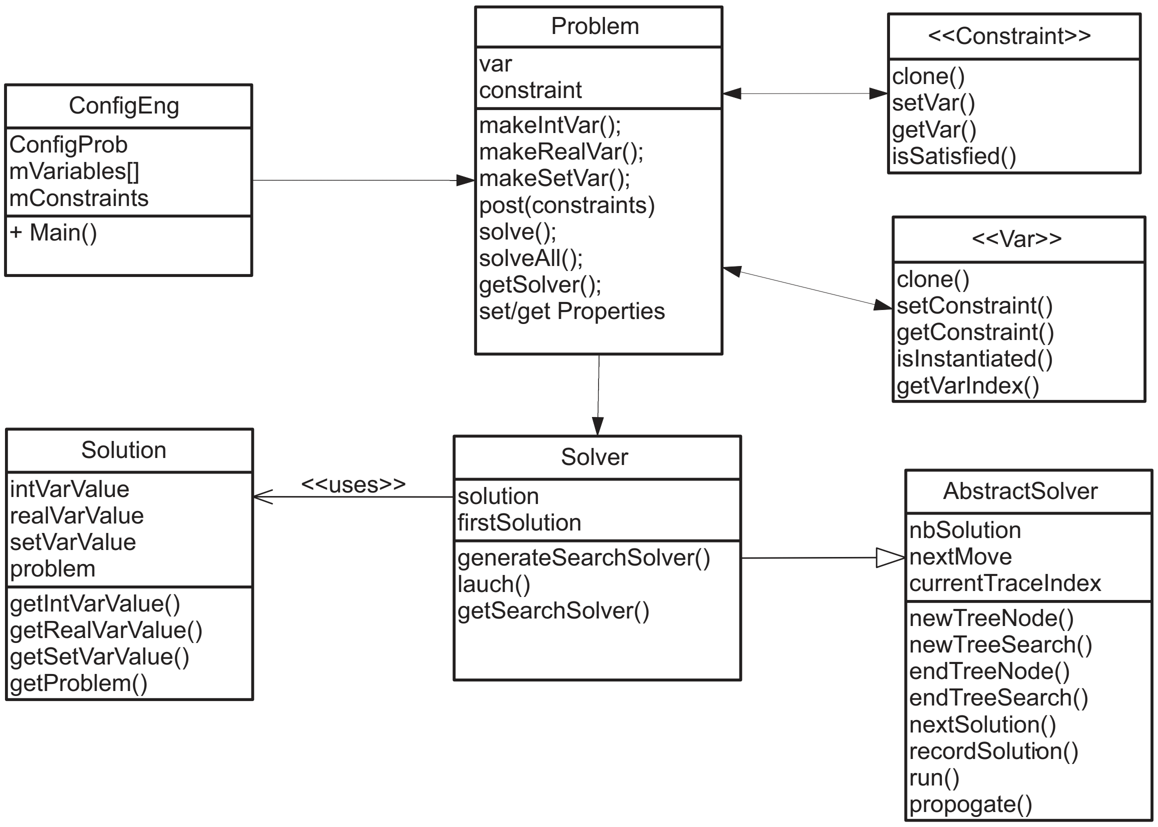


Figure 5.13: UML Diagram of Configuration Solver

In the `problem` class, function `solveAll()` and `solve()` are used to activate the `solver` and launch the CSP solving process with all or single solution respectively. Once solving process finished, function `getSolver()` can be called to trigger the solutions obtained. Class `solver` is instantiated in the main class. It has fields `solution` that record the solutions of the problem. The variable `solution` is instantiated from the `solution` class. It has fields for capturing the values for different types of variables. `GenerateSearchSolver()` is defined to set the search parameters before solving starts. Solving process kicks off when `Launch()` is called. It also inherit from the `AbstractSolver` class. This class is used to define detail parameters for the tree search process. It has functions that define the starting and finishing tree search nodes and paths. Function `propagate()` is called to trigger the Constraint Propagation functions in the library, where different consistency check approaches have been implemented.

5.6 Life cycle Cost Estimation

To implement cost estimation for the product configuration, cost data structure needs to be defined to capture the cost incurred in different activities within the product life cycle. In our work, we have defined cost data model similar to the product data structure. As shown in Figure 5.14, cost estimation module is conducted concurrently with the PDM system. Life cycle cost is modeled into a two layer hierarchy including the cost template and cost data. Cost template is associated with product family architecture, which provides general cost drivers and cost elements without exact data. Product cost data can be populated into the template according to the product configuration selected. For example, two products under the same product family shares the same cost template during the estimation phase, while the output cost data are different.

5.6 Life cycle Cost Estimation

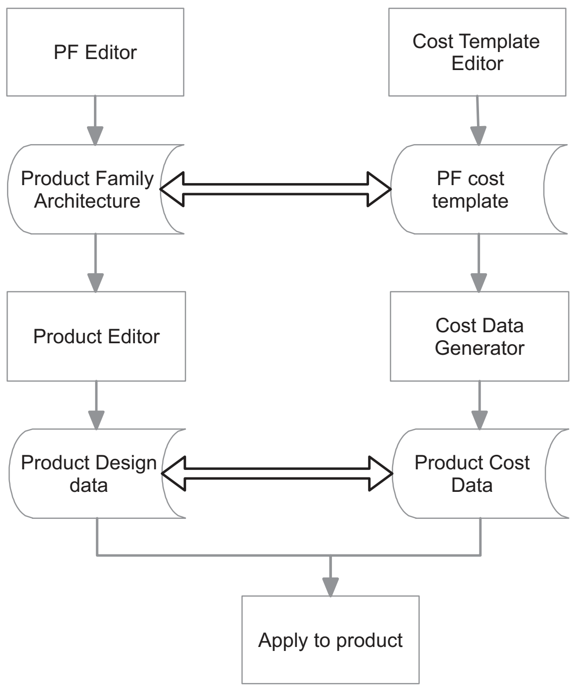


Figure 5.14: Implementation of Lifecycle Cost Assessment with PDM

Figure 5.15 shows the abstract model of product life cycle cost. The cost model consists of four major elements that represents the hierarchy of the life cycle, i.e., a lifecycle can be broken down into different stages, including design, manufacturing, recycle, etc. Each stage consists of different processes, such as machining process and assembly process in manufacturing stage. Each stage contains several activities, such as milling or turning in machining process. Each activity has one or more cost drivers. For example, in milling activity, cost driver is the milling hour and the resource used is labour, machine and raw material. Each of the resource has its quantity measurement for cost. The cost model is designed to adhere to the practical life cycle break down structure. It is proven to be equipped with better cost accuracy.

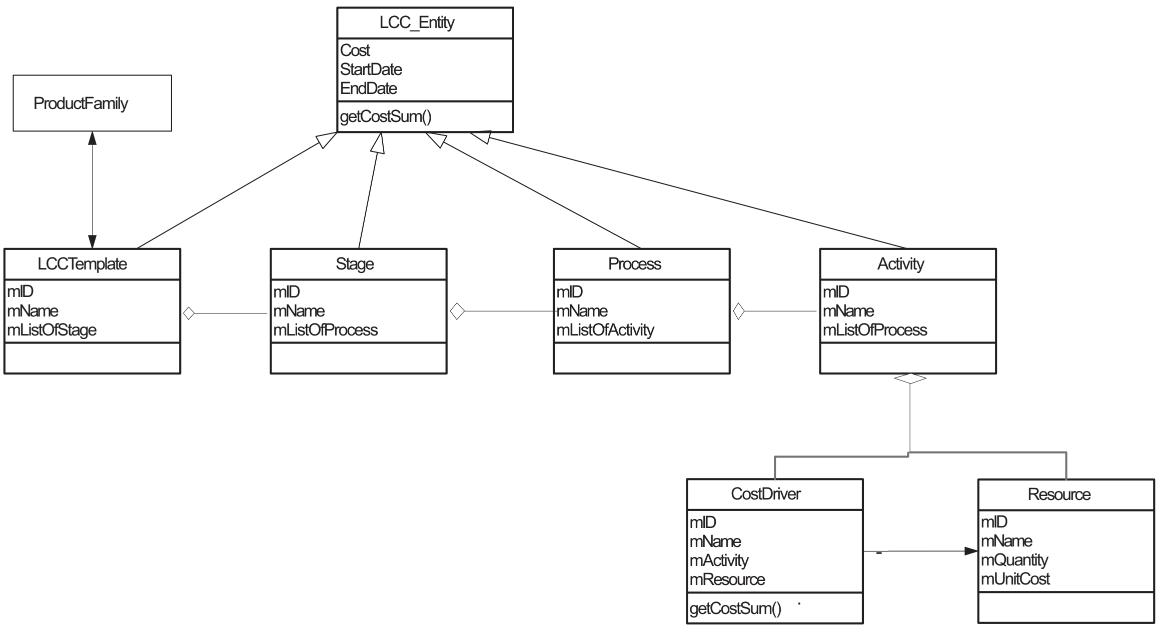


Figure 5.15: UML Diagram of Abstract Lifecycle Cost Model

5.7 Workflow System

To coordinate the whole configuration process, a workflow system is developed to define task sequence and assign users. In order to enhance the adaptability of the system for future development and application, the system is defined as a generic stand-alone application. It allows users to create activities with properties and logic sequence. The output is an XML file that can be used by web-based as well as windows-based configuration application. Figure 5.16 shows the basic GUI of the workflow system. Left panel is the toolbox that contains the elements such as activity entity, logic sequence control element, etc. User is allowed to drag and drop the items from toolbox to the main design panel. In the figure, starting and ending nodes are defined. Activities and linkages are shown in-between. Entity properties can be input from the properties panel on the right. It enables user to assign a personnel to be responsible for each activity. The workflow system is useful for large manufacturing system where tasks are divided into fine details and the number of personnel involved is large. At the initial development of the configuration system, the workflow system has basic main functions, which can be further enriched in the customization for application in client system.

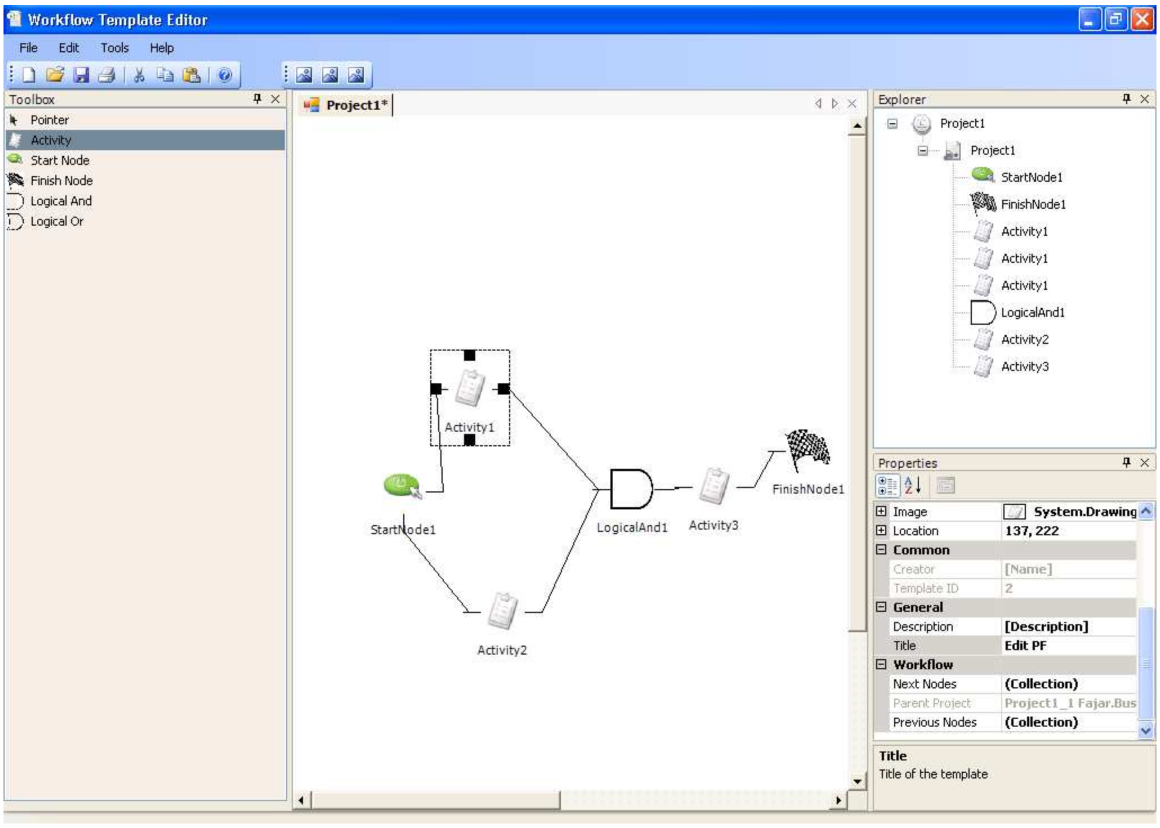


Image	System.Drawing
Location	137, 222
Common	
Creator	[Name]
Template ID	2
General	
Description	[Description]
Title	Edit PF
Workflow	
Next Nodes	(Collection)
Parent Project	Project1_1 Fajar.Bus
Previous Nodes	(Collection)

Title
Title of the template

Figure 5.16: Workflow Editor GUI

5.8 Summary

A constraint-based product configuration system has been developed and presented in this chapter. A system architecture is shown, which consists of main modules including product data management system (PDM), Constraint Manager, lifecycle cost estimator and configuration engine. PDM is developed based on the product data model defined in Chapter 3. Two layers including product family architecture and product data structure has been well developed into the system. Constraint manager is developed to provide interface for user to input design knowledge and store them according to our defined syntax. Configuration solving engine deploys `choco` CSP solver library using the product data and design knowledge to generate feasible solutions. Abstract product lifecycle cost model is presented to model cost based on breakdown activities. Moreover, a generic workflow system is shown to demonstrate the basic feature of the task management and user control for configuration application in large and complicated organization. To summarize, a configuration system has been developed based on the CSP paradigm. Enhancements have been incorporated to increase the value-add of the system.

CHAPTER 6 CONCLUSION AND FUTURE WORK

6.1 Conclusions

Mass customization has been recognized as the trend in the manufacturing sector today. Effective and efficient product configuration tool is the key enabler as it enables companies to generate product variants that meet customer requirements rapidly and provide more product information at the early stages.

In this work, we have given an intensive review on the state-of-the-art configuration methodologies. Constraint-based product configuration approach has been considered to be one of the successful way of modeling configuration problem. We focus on researching into the issues in constraint-based product configuration. A overall framework of the configuration system has been proposed. The software prototype for the product data management has been developed. Our research scope covers the different stages in the configuration, including problem modeling and solving phases.

Product data model that is based on the product family concept has been developed for our application and the product lifecycle cost estimation was brought out to be integrated with the design data model. An enhancement on the traditional product family based data model has been proposed [21] and related configuration algorithm is presented. With the product data model as support, constraint model was constructed to model the design knowledge and the requirement specifications. To ensure consistency between the product data model and constraint model, a direct mapping relationship was defined. We have systematically defined the syntax and semantics of the constraints. A generic CSP solver was deployed to generate solutions based on the defined constraints and product data. By populating data into the product data model and constraint model in the application, the CSP solver acting as basic configuration engine have been developed which is able to generate the solutions that meets the requirements without violating the design constraints. In addition to the proposed approach and system development, we also researched into the factors that affects the constraint solving efficiency. From the literature, one of the factors that affect the efficiency is the variable and constraint ordering in the problem. As configuration problem has strong variable structures, we propose a variable and constraint ordering heuristics to increase the CSP solving efficiency particularly for configuration problem.

On the other aspect of product configuration research, knowledge acquisition is one of the factor that affects a knowledge-based system's performance and acceptance. Manual input of the design knowledge is time consuming and error-prone. We propose an automatic constraint acquisition approach [20] based on existing configuration data. Association rule mining technique is deployed. Experiment has shown that the approach is promising and is able to assist knowledge engineer during the knowledge input process. To realize the constraint-based product configuration capability, a software system consisting of PDM, Constraint Manager, life cycle cost estimator and configuration engine is developed. A basic generic workflow system is developed to control the task sequence and user access for large configuration system. To summarized, the thesis has made the contributions includes:

- A comprehensive study and summary on the existing research work in product configuration.
- Cross-family product data model, a novel product data model, is proposed to enhance the capability in capturing the variety of product data. Product life cycle cost model is proposed to integrate with product data to allow cost estimation.
- Constraint model that is able to enhance the CSP solving efficiency for configuration problem is developed to capture the design knowledge and requirement specifications.
- A novel constraint acquisition approach is proposed and experiment is conducted to demonstrate its effectiveness.
- A software system for constraint-based product configuration is developed. Water heater data from local manufacturer has been successfully tested on the system.
- Other system add-ons including Workflow system and 3D product visualization are implemented to the system.

Although the current system is a success, there are several issues can be looked into in the future work.

6.2 Future Work

- In this configuration system, we focus on the application for OEM manufacturer, where requirements are stated in technical terms. However, for consumer product configuration, requirements are usually given by customers, which is always stated in subjective terms, such as "good quality", "light weight". Therefore, a customer requirements analyzer (CRA) that interprets the requirements into technical terms is necessary to integrate with the configuration system. The challenge is in capturing the variety and ambiguity of customer requirements. Knowledge base is needed for static mapping between requirements and technical terms. For dynamic mapping, machine learning approach may be deployed to discover mapping from existing transactions.

- Although knowledge acquisition has been proposed and experimented to show its significance, it is still in the conceptual stage. Moreover, as product design involves different factors, full automation of the knowledge acquisition is not feasible. Therefore, automation knowledge acquisition can be applied in the process of knowledge input, i.e., it can be used as an assistance to the constraint editor. This in turn helps in the area of human-computer interaction. The challenge is in the system user interface design. Proper integration between knowledge acquisition module and user input needs to be developed. User study has to be conducted to ensure the user-friendliness of the system. On the other hand, data mining techniques can be further researched to enhance the accuracy of the rule generated.

- The current system is a generic configuration system. It can be customized to enhance the feature for specification domain application. The research in this thesis is focusing on the product life cycle based configuration. The configuration activity is within an enterprise. Supply chain management techniques and systems can be researched and integrated with the current system to bring in more user entities. It is to realize the network configuration, which allows more seamless inter-enterprise collaboration and enhances resource sharing. When more people and processes become involved in the configuration process, more sophisticated workflow feature has to be developed. Security of the data of the system will also be a major concern in the practical application. To maximize the system functional sharing and module reuse, the system can be further developed into Web Services for next generation application.

AUTHOR'S PUBLICATIONS

Journal Paper

1. Y. L. Huang, H. Liu, W. K. Ng, W. F. Lu, B. Song, X. Li, "Toward Automatically Generating Constraint-based Configuration Knowledge", to appear in *International Journal of Manufacturing Technology*

Conference Papers

1. Y. L. Huang, H. Liu, W. K. Ng, W. F. Lu, B. Song, X. Li, "Toward Automatically Generating Constraint-based Configuration Knowledge", *International Conference on Manufacturing Automation* (ICMA07), May 2007, [Highly Recommended Papers Award]

2. Y. L. Huang, W. K. Ng, H. Liu, W. F. Lu, B. Song, X. Lin, "Semantic Modeling and Extraction for Cross Family Product Configuration", *International Conference on Service Systems and Service Management* (ICSSSM), Jun 2007

3. H. Liu, Y. L. Huang, W. K. Ng, B. Song, X. Li and W. F. Lu, "Deriving Configuration Knowledge and Evaluating Product Variants through Intelligent Techniques", *Sixth International Conference on Information, Communications and Signal Processing* (ICICS 2007), Singapore, December 2007

REFERENCES

[1] Jess rule language and rule engine. *http://herzberg.ca.sandia.gov*, Accessed in Feb 2007.

[2] Weka, data mining toolkit. *http://www.cs.waikato.ac.nz/ml/weka*, Accessed in Feb 2007.

[3] Choco constraint satisfaction toolkit. *http://choco.sourceforge.net*, Accessed in Jan 2007.

[4] Drool rule language. *http://labs.jboss.com/drools*, Accessed in Jan 2007.

[5] M. Aldanondo, K. H. Hamou, G. Moynard, and J. Lamothe. Mass customization and configuration: Requirement analysis and constraint based modeling propositions. *Integrated Computer-Aided Engineering*, 10(2):177 – 189, 2003.

[6] L. Ardissono, A. Felfernig, G. Friedrich, D. Jannach, R. Schalfer, and M. Zanker. Intelligent interfaces for distributed web-based product and service configuration. *Web Intelligence*, pages 184–188, 2001.

[7] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Schneider. *The Description Logic Handbook: Theory, Implementation and Applications.* 2003.

[8] F. Bacchus and P. V. Run. Dynamic variable ordering in csps. In *Principles and Practice of Constraint Programming - CP '95*, pages 258 – 75, Cassis, France, 1995.

[9] V. E. Barker, D. E. O'Connor, J. Bachant, and E. Soloway. Expert systems for configuration at digital: Xcon and beyond. *Communications of the ACM*, 32(3):298 – 317, 1989.

[10] C. Bessiere, E. C. Freuder, and J. C. Regin. Using constraint metaknowledge to reduce arc consistency computation. *Artificial Intelligence*, 107(1):128 – 148, 1999.

[11] R. W. Bourke. Product configurators: Key enablers for mass customization. *Configuration, A Special Report*, 2000.

[12] Y. Chen. Improving han and lee's path consistency algorithm. In *Tools for Artificial Intelligence (TAI)*, pages 346 – 350, San Jose, CA, USA, 1992.

[13] K. L. Choy, W. B. Lee, C. W. Lau, D. Lu, and V. Lo. Design of an intelligent supplier relationship management system for new product development. *International Journal of Computer Integrated Manufacturing*, 17(8):692 – 715, 2004.

[14] X. Du, J. Jiao, and M. M. Tseng. Product family modeling and design support: An approach based on graph rewriting systems. *(AI EDAM) Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 16(2):103 – 120, 2002.

[15] R. Duray, P. T. Ward, G. W. Milligan, and W. L. Berry. Approaches to mass customization: configurations and empirical validation. *Journal of Operations Management*, 18(6):605–625, 2000.

[16] N. Franke and F. T. Piller. Configuration toolkits for mass customization: Setting a research agenda. *International Journal of Entrepreneurship and Innovation Management*, 2003.

[17] M. Golden, W. R. Siemens, and J. C. Ferguson. What's wrong with rules? In *Proceeding of WESTEX, IEEE Computer Society Press*, pages 162 – 165, Anaheim, CA, USA, 1986.

[18] M. Heinrich and E. W. Jungst. A resource-based paradigm for the configuring of technical systems from modular components. In *Proceedings. Seventh IEEE Conference on Artificial Intelligence Applications*, pages 257 – 64, Miami Beach, FL, USA, 1991.

[19] A. Heylighen and H. Neuckermans. A case base of case-based design tools for architecture. *CAD Computer Aided Design*, 33(14):1111 – 1122, 2001.

[20] Y. L. Huang, H. Liu, W. K. Ng, W. F. Lu, B. Song, and X. Li. Towards automatically generating constraint-based configuration knowledge. *International Conference on Manufacturing Automation (ICMA), Singapore*, 2007.

[21] Y. L. Huang, W. K. Ng, H. Liu, W. F. Lu, B. Song, and X. Li. Semantic modeling and extraction for cross-family product configuration. In *International Conference on Service Systems and Service Management*, 2007.

[22] R. Hull. Mass customization: the new frontier in business competition. *Research and Development Management*, 25(2), 1995.

[23] L. Hvam. A multi-perspective approach for the design of product configuration systems. *http://www.productmodels.org*.

[24] A. Kumar. Mass customization: metrics and modularity. *International Journal of Flexible Manufacturing Systems*, 16(4):287 – 311, 2004.

[25] V. Kumar. Algorithms for constraint-satisfaction problems: A survey. *AI Magazine*, 13(1):32 – 44, Spring 1992.

[26] J. Li. A novel approach to computer-aided configuration design based on constraint satisfaction paradigm. *Master Thesis, Mechanical Engineering, University of Saskatchewan*, 2005.

[27] T. W. Liao, Z. M. Zhang, and C. R. Mount. A case-based reasoning system for identifying failure mechanisms. *Engineering Applications of Artificial Intelligence*, 13(2):199 – 213, 2000.

[28] H. Liu, V. Gopalkrishnan, W. K. Ng, B. Song, and X. Li. Estimating product lifecycle cost using a hybrid approach. *The Second International Conference on Digital Information Management*, 2007.

[29] J. McDermott. R1: a rule-based configurer of computer systems. *Artificial Intelligence*, 19(1):39 – 88, 1982.

[30] D. L. McGuinness and A. T. Borgida. Explaining subsumption in description logics. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence(IJCAI)*, volume vol.1, pages 816 – 21, 1995.

[31] J. W. Meredith and B. M. Kleiner. Empirical design of computer support and staffing in concurrent engineering. *Human Factors and Ergonomics in Manufacturing*, 16(2):177 – 193, 2006.

[32] S. Mittal and F. Frayman. Towards a generic model of configuration tasks. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1395 – 1401, Detroit, MI, USA, 1989.

[33] J. Nanda, H. J. Thevenot, and T. W. Simpson. Product family design knowledge representation, integration, and reuse. In *IEEE International Conference on Information Reuse and Integration*, pages 32–37, 2005.

[34] J. Nanda, H. J. Thevenot, and T. W. Simpson. Product family design knowledge representation, integration, and reuse. *Proceedings of the IEEE International Conference on Information Reuse and Integration*, pages 32 – 37, 2005.

[35] N. Narodytska and T. Walsh. Constraint and variable-ordering heuristics for compiling configuration problems. *IEEE Intelligent Systems*, 22(1):78 – 81, 2007.

[36] H. Nunez, M. Marre, U. Cortes, J. Comas, M. Martinez, I. Roda, and M. Poch. A comparative study on the use of similarity measures in case-based reasoning to improve the classification of environmental system situations. *Environmental Modelling and Software*, 2003.

[37] H. R. Osborne and D. G. Bridge. A case base similarity framework. In *Advances in Case-Based Reasoning. Third European Workshop, EWCBR-96. Proceedings*, pages 309 – 23, Lausanne, Switzerland, 1996.

[38] D. Sabin and E. C. Freuder. Configuration as composite constraint satisfaction. In *Proceedings. Artificial Intelligence and Manufacturing Research Planning Workshop*, pages 153 – 61, Albuquerque, NM, USA, 1996.

[39] D. Sabin and R. Weigel. Product configuration frameworks-a survey. *IEEE Intelligent Systems*, pages 42 – 49, 1998.

[40] K. Saenchai, L. Benedicenti, and R. Paranjape. Solving dynamic distributed constraint satisfaction problems with a modified weak-commitment search algorithm. In *Lecture Notes in Artificial Intelligence*, Utrecht, Netherlands, 2005.

[41] T. Sato. Algorithm for intelligent backtracking. In *Proceedings of RIMS Symposium on Software Science and Engineering*, pages 88 – 98, 1983.

[42] E. Shehab and H. Abdalla. An intelligent knowledge-based system for product cost modelling. *International Journal of Advanced Manufacturing Technology*, 19(1), 2002.

[43] G. D. Silveiraand, D. Borenstein, and F. S. Fogliatto. Mass customization: Literature review and research directions. *International Journal of Production Economics*, 72(1):1 – 13, 2001.

[44] B. Squirel, S. Brown, J. Readman, and J. Bessant. The impact of mass customisation on manufacturing trade-offs. *International Journal of Production and Operations Management*, 15(1):10–21, 2006.

[45] M. Stumptner and A. Haselbock. A generative constraint formalism for configuration problems. In *Third Congress of the Italian Association for Artificial Intelligence, AI*IA*, pages 302 – 13, Torino, Italy, 1993.

[46] N. P. Suh. Design axioms and quality control. *Robotics and Computer-Integrated Manufacturing*, 9(4-5):367–376, 1992.

[47] M. M. Tseng and J. Jiao. Concurrent engineering for mass customization. *Business Process Management Journal*, 4(1):10–24, 1998.

[48] M. M. Tseng and J. Jiao. Mass customization. *Handbook of Industrial Engineering, Technology and Operation Management*, 2001.

[49] C. M. Vong, T. P. Leung, and P. K. Wong. Case-based reasoning and adaptation in hydraulic production machine design. *Engineering Applications of Artificial Intelligence*, 15(6):567 – 85, 2002.

[50] I. F. Weustink, E. Brinke, A. H. Streppel, and H. J. Kals. Generic framework for cost estimation and cost control in product design. *Journal of Materials Processing Technology*, 103(1):141 – 148, 2000.

[51] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. 1999.

[52] K. Wolter, L. Hotz, and T. Krebs. Model-based configuration support for software product families. *Mass Customization: Challenges and Solutions*, 2006.

[53] H. Xie, P. Henderson, and M. Kernahan. Modelling and solving engineering product configuration problems by constraint satisfaction. *International Journal of Production Research*, 43(20):4455 – 4469, 2005.

[54] X. Xu, J. L. Chen, and S. Q. Xie. Framework of a product lifecycle costing system. *Journal of Computing and Information Science in Engineering*, 6(1):69 – 77, 2006.

[55] W. Zhang and R. E. Korf. Depth-first vs. best-first search: new results. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 769 – 775, 1993.