A Novel Approach to Computer-Aided Configuration Design Based on Constraint Satisfaction Paradigm

A Thesis Submitted to the College of

Graduate Studies and Research

in Partial Fulfillment of the Requirements

for the Degree of Master Science

in the Department of Mechanical Engineering

University of Saskatchewan

Saskatoon

By

Jingxin Li

Permission to Use

In presenting this thesis in partial fulfilment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Mechanical Engineering

University of Saskatchewan

Saskatoon, Saskatchewan (S7N 5A9)

ABSTRACT

The trend in today's manufacturing industry is changing from mass production to mass customization. The companies which win the markets are those which can deliver highly customized products at the fastest rate and allow for life-cycle participation of customers regardless of where they are and when they participate. One of the strategies for implementing the mass customization paradigm is to implement the product development according to the assemble-to-order (ATO) pattern. Under the ATO pattern, the design of a product becomes the determination of a configuration which contains a set of pre-developed components – configuration design for short. The configuration design problem can be well treated as a constraint satisfaction problem (CSP). The mature methods are available for CSP, but there are several limitations with CSP for configuration design.

This thesis proposes a novel approach to configuration design. This approach is based on a CSP but adds a wrapper (product data model, PDM for short) over the CSP model. Consequently, both the customer and the other life cycle development programs only communicate with the PDM, and a more intelligent and user-friendly computer system for configuration design can then be implemented. Both the conceptual design and implementation of such a wrapper are discussed in this thesis. A computer prototype system for elevator design is developed for demonstrating the effectiveness of this approach.

ACKNOWLEDGMENTS

Upon the completion of this thesis, I would like to express my sincere gratitude and appreciation to my supervisor Professor W.J. (Chris) Zhang for his invaluable guidance, constructive discussion, inspiration and encouragement, without which this research work would not be carried out successfully.

My appreciation is extended to the members of the Advisory Committee: Professor S. Habibi, Professor I. Ogoucha, for their scholarly suggestions, advices and examination in the period of this research. My appreciation also goes to H. Xie and Concurrent Engineering Research Group in IMTI, National Research Council (NRC) for their scholarly and partial financial support. My appreciation would also go to Loken Engineering Services for its resourceful working environment and generous financial support to sustain my expenditure.

I am grateful to my dearest daughter for her understanding to the countless late night study, and to my absence for many of her school activities. I am also greatly indebted to my parents, my grandmother, my siblings, and other family members for their continued moral support and love.

This research was made possible by the financial support from the National Sciences and Engineering Council (NSERC) of Canada.

Dedicated to

My dearest daughter Sophia (Song) Ge

And

My parents,

Mr. M.S. Li and Ms. Z.F. Yang

For their continuous love and unwavering support!

TABLE OF CONTENTS

List of Figures

List of Abbreviations

AC:	Arc Consistency
API:	Application Programming Interface
ATO:	Assemble-to-Order
BOM:	Bill of Material
CAD:	Computer Aided Design
CAM:	Computer Aided Manufacturing
CASE:	Configuration Design Knowledge Base
CIM:	Computer Integrated Manufacture
CRM:	Customer Relationship Management
CSP:	Constraint Satisfaction Paradigm, Problem, Programming
DBMS:	Database Management System
DTD:	Document Type Definition
EDI:	Electronics Data Interchange
ERP:	Enterprise Resource Planning
ETO:	Engineer-to-Order
JCL:	Java Constraint Library
JSP:	Java Server Page
HTML:	Hyper Text Markup Language
LCCD:	Lifecycle Configuration Design
MRP:	Manufacture Resource Planning
PC:	Path Consistency
PDM:	Product Data Model, Product Data Modeling

PDM-CSP: Integrated PDM and CSP

PLM: Product Lifecycle Management

POD: Product Ontology Dictionary

Prob-CSP: Probabilistic CSP

SCM: Supply Chain Management

SGML: Standard Generalized Markup Language

STEP: Standard exchanging protocol

STO: Stock-to-Order

UML: Universal Model Language

XHTML: E**X**tensible **H**yper**T**ext **M**arkup **L**anguage

XML: eXtensible Markup Language

Chapter 1
Introduction

1.1 Evolution of Information Technology to Manufacturing

Manufacturing has undergone a long process of awareness of the strong positive impact of information technology to manufacturing. Two decades ago, the paradigm popular in manufacturing is computer integrated manufacture (CIM). The CIM paradigm leads to a great improvement of manufacturing practice but at some sacrifice as a result of high expense. With the development of the internet technology, communication between two remote ends is greatly facilitated. The organization of a manufacturing firm becomes virtual in the sense that each manufacturing unit keeps competitive components within themselves. Manufacturing activities become more of a "networking activity".

The trend of today's manufacture industry is changing from mass production to mass customization. The companies which win the markets are those which can deliver highly customized products with the fastest speed. As such, the production development changes from "stock-to-order", to "assemble-to-order", and/or to "engineer-to-order".

Stock-to-order (STO) refers to a manufacturing situation where a whole product (e.g., a computer) is available in the manufacturer's inventory. For instance, if one wants to purchase a laptop, one goes to the computer shop or manufacturer and gets one provided that the

manufacturer has prepared computers in its inventory (i.e., the manufacture adopts the stock-to-order pattern).

Assemble-to-order (ATO) refers to a manufacturing situation where the product structure is known. The product structure means (1) the number of component types (or components for simplicity), (2) how these components are connected. ATO manufacturing process is then to determine instances of the components to make an assembly to meet the customer's requirement. ATO differs from the stock-to-order in that in the case of ATO, instances of components may not be available to the manufacturer which directly communicates with a customer, and they are usually supplied by other manufacturers. ATO is often integrated with the order, supply, and production systems so that once a product is configured, delivery can follow immediately. Usually, the engineering work is not required for ATO pattern. However, some manufacturing works may be needed where a product is assembled.

Engineer-to-order (ETO) deals with problems where not all components are ready to use; some may need to be designed and then fabricated specifically to meet customer's requirements.

This thesis concerns product development which follows the assemble-to-order (ATO) pattern; specifically the development of a computer support system for ATO pattern.

1.2 Product Data Modeling

The key technology to develop an effective computer support system for product development is Product Data Modeling (PDM). From the point of view of modeling, the PDM is a process of

establishing a data model that represents information and knowledge generated during a product development life cycle. From the point of view of management, the PDM is a process of managing data. Database technology is a powerful tool for PDM. Database technology provides the notion of data model [Codd, 1970] and the notion of semantic data model [Chen, 1976]. A data model contains a set of rules to define the structure of, the constraint of, and the valid operation on data. Quite often, the applications of a data model focus on the structure and constraint. A semantic data model focuses on the meaning (or semantics) of data in the context of a discourse of an application [Zhang, 1994]. In other words, a semantic model concerns with what information or knowledge a series of symbols asserts. A semantic data model is therefore also called "conceptual data model". This thesis research focuses on conceptual data modeling; specifically on what information is needed to support the assemble-to-order (ATO) pattern. The computer system that supports the assemble-to-order pattern can then be designed to capture this information. There will be a further discussion in Section 1.7 regarding the relationship among a conceptual data model, its implementation, and its application.

1.3 Product Configuration Technology

The most commonly used definition of the configuration task was given by Mittal & Frayman [1989]: *The configuration of an artefact is a set of interconnected components that are chosen from predefined sets of component types called the catalog of component types. Specifically, a component is described by a set of properties, ports for connecting it to other components, constraints at each port that describe the components that can be connected at that port, and other structural constraints.*

An augmented configuration is defined as the configuration (as defined previously) together with the background about why the configuration is needed and/or the rationale about how the configuration is determined.

Configuration design is a process of determining one or more configurations that satisfys the function and constraint requirements. It should be noted that in literature, there is a notion called "configuration management" [Lyon, 2000]. The configuration management mainly concerns with the change management, and it addresses the following issues: who suggests changes, who assesses impact of the changes, and who approves the changes. Configuration management is thus different from configuration design.

1.4 Research Questions

Question 1:

What should be a computationally effective representation of a product configuration for the Assemble-to-Order product development?

In current literature, the answer to this question appears to be (1) the expert system (rule based system), or (2) the database approach. The main rational for the expert approach is that the expert system is about capturing the constraints in a configuration using the syntactic expression – rules. But the expert system is known for its inflexibility with respect to the change of requirements and knowledge, or to the maintenance of the system. The database approach, which was pioneered by Zhang [1994], is indeed very general with respect to information representation, but not computation-oriented. This means that in order to design a configuration,

information needs to be extracted from a database and converted into a form on which algorithms can be applied. An alternative solution is Constraint Satisfaction Problem (CSP) [Tsang, 1999; Miguel and Shen, 2001]. The CSP is a paradigm or an approach to represent a problem or a system into formalism called constraint, and the solution to the problem satisfies the constraint. The CSP is not only highly declarative to represent design knowledge, but also domain independent. This thesis adopts the CSP approach to Question 1. Chapter 2 will present details of the CSP approach. Specifically, one will see some unresolved problems with CSP itself and the application of CSP to configuration design. This thesis will address these problems and present a new approach which integrates PDM and CSP.

It is clear that the answer to this question will eventually come to a system that determines a configuration, given a set of requirements; such a system is also called configurator in literature.

Question 2:

Suppose that the CSP approach is taken to build a configurator. The knowledge representation will be the one suitable for computation, but not efficient for representing information that sits at the back end, e.g., the rationale for a design decision. The question is then as follows:

What is the role of the CSP configurator in the context of mass customization manufacturing paradigm?

The answer to this question has not been found. The question reads like a traditional question called integration, yet in the new context which is characterized by (1) the configuration technology, and (2) the internet technology.

This thesis undertakes to answer the two questions above. The next section provides a literature review of relevant studies.

1.5 Configurator: a Critical Review

Configurator was originally designed as an interface on the top of the ERP (Enterprise Resource Planning) system. This is especially introduced to facilitate the acquisition of products from customers with a focus on the price and deliverable time. The commercial configurators, such as PTC's Windchill™ Product and FirePond's Sales Performer™, are successful implementations of the configurator concept; in addition, they usually provide the customer with access to their systems at any time and any place through the internet technology. This kind of configurators may be called the ERP-based configurator.

One of the essential assumptions underlying the ERP-based configurator is that products are pre-designed. Specifically, components are ready to go, and they just need to be assembled [Tiihonen et al., 1996; Tiihonen and Soininen, 1997]. The ERP-based configurator cannot work for the situation where the product needs some engineering and manufacturing work. For example, the customer may prefer to a color of the interior of the elevator, which is not available in the current component repository. In this case, the elevator manufacturer may reject the customer's request

for that special color, but the customer oriented manufacturing practice would try to tailor whether the component with that color could be painted and produced, which may need to contact the manufacturer's supplier or sub-contractor who does the painting job for the interior. Another example is such that the customer may want to install a video camera in a place where the existing elevator is designed not to hold the camera in that place. To accommodate that requirement from the customer will require a little bit of engineering work to assemble a product.

It might be quite true that the configurator without the capability of accommodating engineering and manufacturing works (more or less) will considerably compromise the manufacturer's philosophy: customer-oriented product development. Mesihovic and Malmqvist [2000] suggested a PDM integrated configurator. Their main idea was to have the notion shown in Fig. 1.1. This notion was primarily based on the observation that PDM is supposed to support all engineering works (see the discussion in Section 1.3). This idea is promising; yet they have not given details of their system.

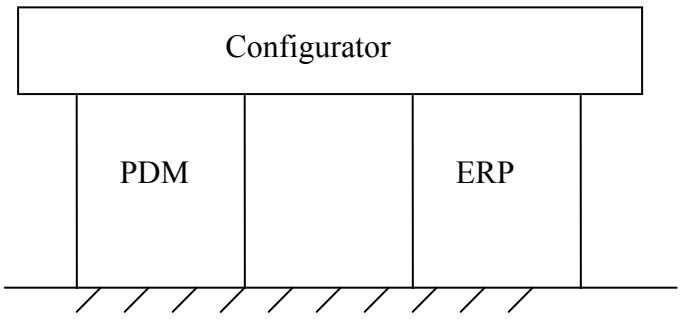


Figure 1.1 The Notion of a PDM Integrated Configuration

With respect to the configuration system by itself, the following issues are of concern. The first issue is whether a configuration system supports configuration design where topology of a

configuration is changed during the process of configuring product. Note that, this issue has gone beyond the ERP-based configurator. At this point, the CSP-based configurator solver has made this possible [Mittal and Falkenhainer, 1990]. Mailharro [1998] commented on the work by Mittal and Falkenhainer [1990], saying that this work sets a limit (maximum) on the number of variables allowed, and questioning the applicability of this approach for large configuration problems. The second issue is the knowledge representation of the product configuration. There are basically three requirements on such a knowledge representation: (1) the flexibility in the sense that knowledge can be easily maintained, (2) the expression power for representing semantics, and (3) the efficiency in terms of the computational time.

To meet requirement (1), the CSP knowledge representation formalism is a must. To meet requirement (2), there are several generic semantics for configuration design, such as multiple occurrences of components [Stumptner et al., 1998]. Usually, CSP will have to rewrite the constraints many times to match the times the component is used. Bowen and Bahler [1991] used a language based on the semantics of free logic to address the multiple occurrence problems. To meet requirement (3), as well as requirement (2), a configuration as composite constraint satisfaction was proposed by Sabin and Freuder [1996], which extended the standard/conventional CSP to accommodate issues such as unknown *a priori* number of constituent parts of a system. The knowledge is expressed as a hierarchical structure. The domain of a variable is a set of entire sub-problems with their own variables and constraints. After instantiating a variable with one of the possible sub-problems, the variables and constraints of the sub-problem are added to the constraint network.

In general, the current configurtors, including both research and commercial based ones, can only meet the functions which were described by Mittal and Frayman [1989]. They need to be extended in the aspects of (1) optimization, (2) user preference, and (3) integration of product configuration with many other engineering and manufacturing systems.

1.6 Research Objectives

Objective 1:

Extend a CSP representation for explicitly incorporating the customer preferences and composite object constraint.

Objective 2:

Develop an integrated PDM and CSP approach to configuring products.

Objective 3:

Develop a framework for integrating a configurator with other product life cycle development systems

The particular activities in the product life cycle considered are the engineering design, and parts acquisition and supply. The process planning is not the scope of this thesis, which is considered as an internal business process in a partner company. Furthermore, the notion of framework implies that the question of what information is needed will be the focus. The implementation is merely for the purpose to give impression of what a system could achieve based on the concept and methodology developed and thus to enhance understanding of the concept and methodology.

1.7 General Research Method

1.7.1 General System Development Approach

As mentioned previously, this thesis takes product data modeling as a main approach. The strategy for data modeling is to follow the ANSI/SPARC database architecture [Date, 1990], i.e., the conceptual view, the internal view, and the external view (see Fig. 1.2). The conceptual view of data or database model answers the question: what information is needed for a discourse of an application. The internal view, however, answers how to implement the model resulted from the development at the conceptual view. The external view determines the aspect of the model at the conceptual view for a particular need of an application under consideration. This thesis focuses on the conceptual data modeling, as also mentioned before. Universal Modeling Language (UML) [Booch et al, 1999] will be employed for development of a conceptual view or model for particular applications.

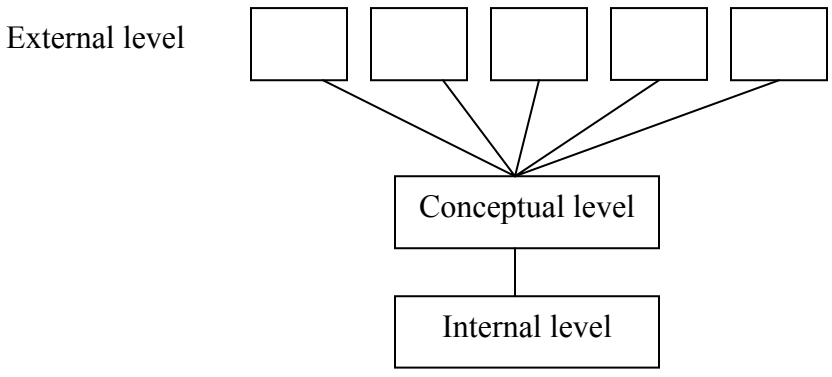
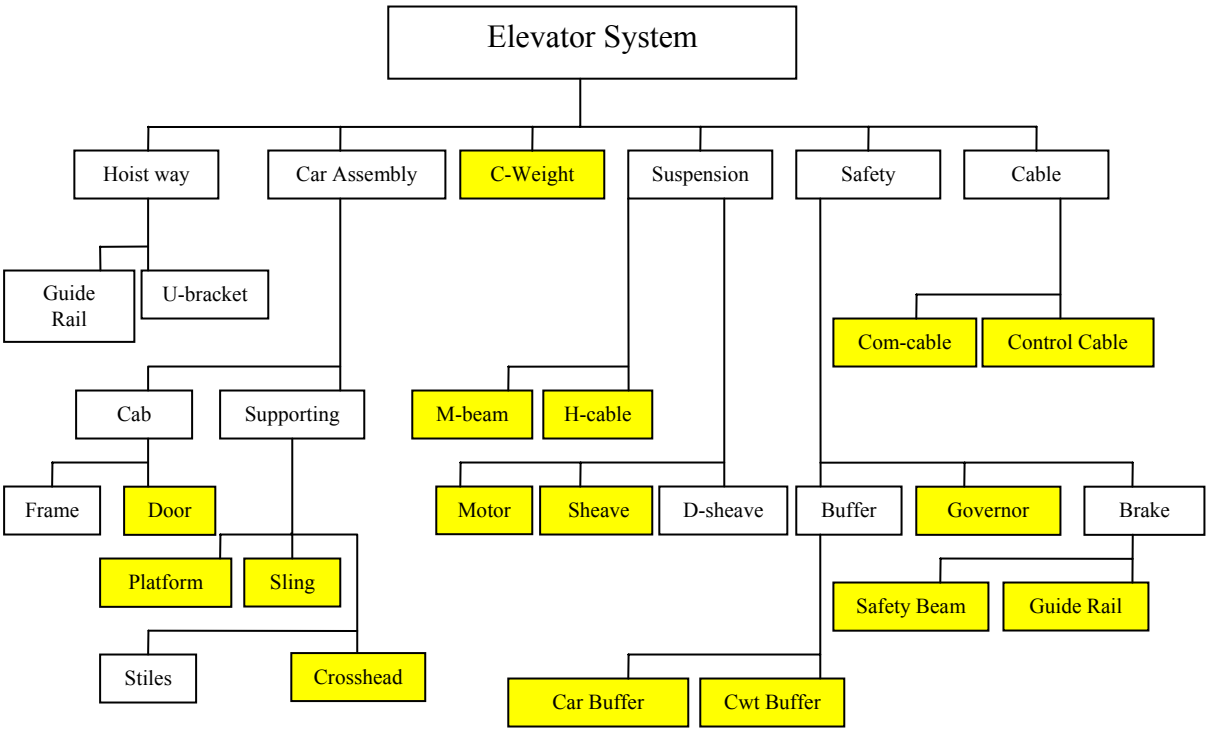


Figure 1.2 Three Levels of Architecture

1.7.2 Case-based elaboration

Another general methodology for this research is case-based presentation method. The case-based method is to take a case or example for elaborating and establishing hypotheses, developing models, and testing the hypotheses through the evaluation of the models. Throughout the thesis, an example regarding an elevator system is taken.



Note: The blocks shaded are the components considered in this thesis

Figure 1.3 System Decomposition of the Elevator System

An elevator system consists of 16 components [Yost, 1996], and they are, respectively, Door, Platform, Sling, Safety, Crosshead, Car Buffer, Cwt-Buffer, C-Weight (counter weight), M-beam (machine beam), Motor, Sheave, H-cable (Hoist cable), Com-cable, Control Cable, Governor, Safety Beam, and Guide Rail. There are four additional components which are optional: Car Lantern, Car Level Indicator, Car Phone, and Car Communication. Their relationships can be visualized by the assembly structure shown in Fig. 1.3. For each component, there could be several alternatives or instances. The selection of their alternatives in order to achieve a desired performance at the whole system level makes sense for the elevator system design to be a configuration design problem.

1.8 Organization of the Thesis

The remainder of this thesis is organized as follows: Chapter 2 presents a literature review of CSP, where there is a discussion of limitations of CSP. Chapter 3 proposes a new approach to configuring products by integrating PDM and CSP. This new approach may also be viewed as an enhancement of CSP to make it more semantic in addition to its powerful computation framework. Furthermore, the preference and composite object problems are addressed in this chapter. Chapter 4 presents some ideas about how to bring the configurator into a product life cycle development. Chapter 5 presents some implementation works to demonstrate the effectiveness of the ideas proposed in the preceding chapters. Chapter 6 is a conclusion with future work and recommendation.

Chapter 2
Constraint Satisfaction Programming (CSP)

2.1 Introduction

In the mid-80s, constraint satisfying programming (CSP) was developed as a computational technique, which is a result of combining artificial intelligence and computer programming techniques. CSP promises to provide solutions to some NP problems and scheduling problem. This chapter serves as a brief introduction to CSP. In Section 2.2, the definition of CSP is presented, and the mathematical model of CSP is defined. Section 2.3 presents some solving methods for the CSP problems. Optimization is an important part, or extension to the original CSP, and it will be discussed in Section 2.4. Section 2.5 provides a broad view of CSP extensions. Section 2.6 discusses the so-called preference problems in CSP. Section 2.7 discusses some limitations with CSP problems.

2.2 Definition of CSP

Definition 1: A CSP is a triple $P = \{V, D, C\}$, where

- $V = \{v_1, v_2, \ldots, v_n\}$ is the set of variables called the domain variables;
- $D = \{D_1, D_2, \ldots, D_n\}$ is the set of domains. The domain is a finite set containing possible values for the corresponding variables;
- $C = \{c_1, c_2, \ldots, c_n\}$ is the set of constraints. A constraint c_i is a relation defined on a subset of $\{v_i, \ldots, v_k\}$ of all the variables; that is, $\{D_i, \ldots, D_k\} \supseteq c_i$.

If a constraint c_i is defined on a set that only has one or two elements, this constraint is called *unary* or *binary* constraint, correspondingly. The remainder of constraints are called non-binary constraints. A CSP is a binary CSP if all its constraints are *unary* or *binary*. Non-binary constraints can be transformed into several equivalent binary CSPs. Therefore, only binary constraints are considered in this thesis.

The structure of a binary CSP may be represented by a constraint graph, which is defined as follows: variables are represented with nodes, and the constraints between them are represented with edges. The labels of the edges represent the constraints and the labels of the nodes represent the domain of the variables (see Fig. 2.1). In Fig. 2.1, there are nodes: A, B, C, D, and E. The digits in the parenthesis behind the node indicate their domains. A node could have edges directing to the node itself, which implies the unary constraint (in Fig. 2.1, node B). The generic format for the expression of a binary constraint is as follows:

Binary constraint: = operand 1| operator| operand 2

For example, if A and B are two variables, and A is greater than B, the binary constraint for this can be expressed as

A>B

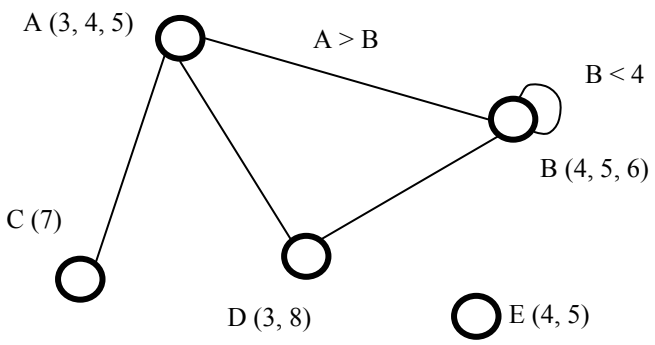


Figure 2.1 CSP Graph Representation

Definition 2: Assignment: It is a mapping from a set of variables to their corresponding domains.

Let v_i be a variable and D_i its domain. The process that v_i takes a value, say d_i from the domain D_i is called assignment. Such an assignment is denoted (v_i, d_i).

For a CSP problem which has a set of variables, say $v_1, v_2, \ldots, v_m$, the assignment for all the variables is denoted $\{(v_1, d_1), (v_2, d_2), \ldots, (v_m, d_m)\}$. When all the variables are assigned a value, the assignment is called complete, otherwise partial. For a complete assignment, one may write as $\{d_1, d_2, \ldots, d_m\}$. The expression $\{d_1, d_2, \ldots, d_m\}$ is also called the *value tuple*. The set of all possible complete assignments is called the assignment space. This space could be very large, depending on factors, such as the number of variables, size of domains, and tight or loose constraints. The following is an example to illustrate the assignment:

Suppose that there are three variables A, B, C and each of them has a domain as follows:

D_A: {1, 2, 3, 4} D_B: {5, 6, 7} D_C: {8, 9, 10, 11}

Each variable is assigned a value from its domain as follows:

A=1, B=6, C=9

Assignment to variables can be represented as {(A, 1), (B, 6), (C, 9)}, or more commonly as {1, 6, 9}. This is also called the value tuple. An incomplete assignment (for example, without assignment to variable C) may look like {(A, 1), (B, 6)}, or {1, 6}.

Definition 3: Consistent assignment: Constraint is satisfied in assignment if each of its variables gets a value such that the value tuple satisfies all constraints.

Definition 4: Complete or partial consistent assignment: A partial consistent assignment refers to the assignment that satisfies a partial set of constraints. A solution to a CSP problem is a complete consistent assignment which means that all constraints are satisfied.

Definition 5: Over constrained: If for a problem there is no such a complete assignment, the problem is called over constrained or inconsistent problem. Correspondingly, a CSP problem with more than one solution is called under-constrained problem.

Definition 6: Relaxing or tightening of constraint: Given a set of constraints, relaxing of constraints means that one or more constraints are removed from this set; while tightening of constraints means to add one or more constraints to this set.

2.3 Methods for Solving a CSP Problem

Finding a consistent assignment to all variables of a CSP problem is the process to solve a CSP problem. Basically, there are two kinds of strategies to find solutions [Tsang, 1999]: systematic

search and repair methods. Systematic search assigns consistent values to variables one by one through a systematic way. Repair method assigns values randomly regardless of constraints, and then repairs the assignments that violate constraints. Repair strategy is often called heuristic strategy or stochastic strategy [Reeves, 1991]. Repair strategy could be ineffective for some large size and tight constraint problems. In this thesis, only systematic methods were applied, which are backtracking and backjumping. In addition, the consistency technique was used to pre-process the variable domains for improving the efficiency of a solving process.

2.3.1 Backtracking

The backtracking solving strategy assigns values to particular variables and extends a partial assignment incrementally. Each time a variable is instantiated, constraints associated with this variable are tested. If some of the constraints are violated, the variable is reassigned a new value from its domain until a consistent assignment is found. If none of the values in the domains are found to form a consistent assignment, the algorithm will turn to the nearest point at which a consistent assignment was established. An example helps to illustrate this method.

Suppose there are variables X, Y and Z with constraints $X \neq Y$, $Y \neq Z$, $X \neq Z$:

Variables and domains: D_X {t, e, f}
D_Y {t, e, f}
D_Z {t, e, f}

The backtracking method for this CSP problem is illustrated in Fig. 2.2.

Under the systematic searching approach, different heuristics can be used to improve the efficiency. It is generally true that variable and value orders are critical for efficiency of problem

solving. For example, in 8-queen problem [Marriott and Stuckey, 1998], the fail-first heuristic is used to determine which variable needs to be instantiated/assigned first. The fail-first rule says that the variable which has smaller domain should be reassigned first. This is because the dead end would be found earlier. Values can be ordered either as ascending, descending sequence, or other heuristics. There will be a discussion about value and variable ordering later in this chapter.

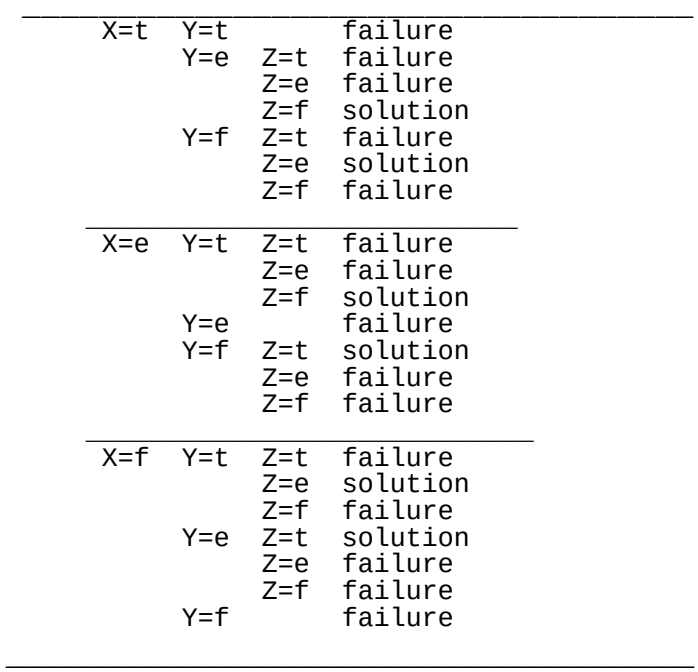


Figure 2.2 An Example of Backtracking

There are three major drawbacks of the standard backtracking method:

- Thrashing, i.e., repeated failure due to the same reason;
- Redundant work, i.e., conflicting values of variables are not remembered, and
- Late detection of the conflict, i.e., conflict is not predicted before it really occurs.

Two methods, backjumping and backmarking are available for addressing the first two drawbacks. The consistency technique solves the third problem.

2.3.2 Backjumping

This is a method to avoid thrashing in the Backtracking method. The process of backjumping is exactly the same as backtracking, except for their different strategies for looking back. When any constraint is violated, the backtracking method will take action of moving one step back, while the backjumping method will first analyze the sources of inconsistency, and then jump to the source point where inconsistency comes.

2.3.3 Consistency Check

Consistency check is a pre-processing of solving a CSP problem, which is also called, in different literatures, local consistency, consistency enforcing, constraint propagation, filtering, or narrowing algorithms [Apt, 1999]. The objective of consistency check therefore is to eliminate those values from the domains of the variables, which do not meet any constraint. The consistency check helps to prune the search tree dramatically in many cases and leads to a smaller space to search. There are three types of inconsistency checks:

Node-consistency check: It removes any value that does not meet the unary constraint from the domain of a variable.

Arc-consistency (AC) check: It deals with consistency between two variables e.g. (A, B). Specifically the rule corresponding to the arc-consistency check is such that a constraint is arc consistenct if for any value in the domain of A in this constraint there is a value in the domain of

another variable such that the constraint is satisfied. CSP is arc consistent if all the constraints are arc consistent. The simplest algorithm for achieving arc-consistency is to repeat revising. An example of arc-consistency is shown in Fig. 2.3. In Fig. 2.3, the constraint Y=2X is a binary constraint on domain Y and domain X. Since the largest value in domain Y is 10, the values over 5 in domain X should be crossed off for any valid assignment based on the constraint Y=2X. So only five values (less than or equal to 5) are left in domain X. Therefore domain Y can be pruned to include only five values that are double of the values in Domain X. A further inference is such that values in Domain X is modulo number, so the values in Domain X can be further reduced to three numbers: (1, 3, 5). Finally the values in Domain Y are: (2, 6, 10). This algorithm is called AC-1, and it suffers from the problem of non-necessary repetition of revisions. There are some other AC algorithms, named AC-2, AC-3, until AC-7, details of which refer to [Bartak, 2001].

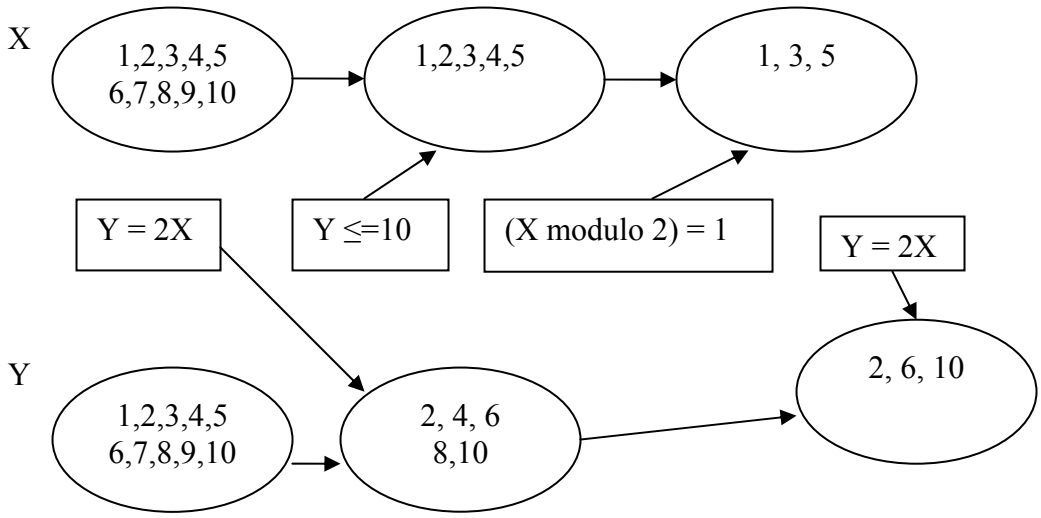


Figure 2.3 Illustration of Constraint Propagation for Arc-Consistency

Path Consistency (PC) check: A path is consistent if for every pair d_1, d_n of consistent values (i.e., this pair satisfies all binary constraints between V_1 and V_n) there exist values d_2, …, d_{n-1}

such that all the constraints between d_i, d_{i+1} are satisfied. CSP is path consistent if all paths are path consistent.

A problem does not necessarily have a solution when it is AC consistent. In contrast, path consistency assures that a path consistency CSP must have at least one solution if none of the domains are empty. From this sense, PC is said to be stronger than AC. However, PC is rarely used in practice because of its computational complexity and high demand on computer memory.

K-consistency check: Node, arc, and path consistency are instances of a general notion called k-consistency. CSP is k-consistent if every consistent (k-1)-tuple can be extended to a consistent k-tuple.

2.3.4 Several Heuristics Rules for Searching

Three heuristics rules are introduced here, the value order, the variable order, and the pseudo solution. The value ordering rule is such that values in domains are arranged in a certain order according to a certain attribute which is meaningful to a particular application problem. In engineering and manufacturing applications, the attribute can be such as cost, performance, weight, and delivery time.

Variable ordering rule is to organize the variables based on some criterion. One of such rules is to order the variable based on the number of partnerships of a particular variable with other variables. For example, if there are five variables: A, B, C, D, and E. A has constraints with B, C,

and D, B with D and A, C with A, D with A and B, and E with none. In this case, the numbers of participations for A, B, C, D, and E are, respectively,

A is 3, B is 2, C is 1, D is 2, and E is 0.

The order of the variables, descending in this case, will be then A, B, D, C, and E. This way is conductive to detect unresolved branches at the early stage of search.

The variables can also be ordered in terms of their importance with respect to a particular application problem. For example, a variable which "contributes" most to the performance may be arranged on the top.

The main idea of the pseudo solution rule is to predict as early as possible whether a search along a particular branch of a tree is promising. The unpromising branch would be abandoned. This is achieved by introducing a constraint into the searching space. Suppose that an initial solution is found, denoted S_0. A constraint is established as follows:

$$S_{i+1} < S_i,\ i = 0, 1, 2, \cdots \cdots$$

where S_i is the solution. The operation '<' represents the preference suggested on the solution. In the case of product configuration design, such preferences may be: the cost is lowest; the weight is lowest, etc.

The pseudo solution rule may be combined with value ordering rule. For example, the cost is the criterion to order the value. Then, values in each domain are ordered in terms of a descending or ascending order. In this case, S_0 can be obtained simply by picking up the first value in all domains subject to other constraints.

2.4 Constraint Optimization

In many real life problems, the interests are probably only on a certain solution, instead of all solutions or a solution obtained at first. The quality of a solution is measured by an application dependent function called the objective function. The goal is to find such a solution that will minimize or maximize the objective function with respect to the applications. This is called CSP optimization. The definition of a CSP optimization problem is given as follows:

(1) A tuple of n variables, $V = (v_1, v_2, \ldots, v_{k,}\ v_{k+1,}\ \ldots, v_n)$, where variables 1 to k measure the multi objectives to be optimized, and variables k+1 to n are special variables without any sub objectives.

(2) A tuple of n domains, $D = (d_1, d_2, \ldots, d_n)$, such that $v_i \in d_i$.

(3) A set of constraints among variables, $R = \{r_1, r_2, \ldots, r_m\}$, that restrict the domains of the variables.

(4) An objective function, z(V), which is minimized or maximized.

The CSP optimization problem consists of a standard CSP problem and an objective function. There are two strategies for solving a CSP optimization problem, namely, the standard search and the dichotomic search methods.

2.4.1 Standard Search

The most widely used algorithm for finding an optimal solution is called Branch and Bound (B & B) method, also called standard search [Lustig and Puget, 2001].

The (B & B) searching procedure is first to find a feasible solution, while ignoring the objective function $z(x_1, x_2, \ldots, x_n)$ (x_i is the value of the variable). Let $(y_1, y_2, \ldots, y_n)$, where y_i is the value of the variable, represent such a feasible solution. The search space is then pruned by adding a new constraint

$$z(y_1, y_2, \ldots, y_n) > z(x_1, x_2, \ldots, x_n)$$

to the system. After that, the search continues. The added constraint specifies that any new feasible solution must have a better objective value than the current point, giving a new lower bound. Consistency or propagation of this constraint may cause the domains of the variables to be reduced, and thus reduce the size of the search space. The search procedure concludes until no feasible solution is found. The last feasible solution is then taken as an optimal solution. The exhaustive search is effective for problems where the searching space is not too big.

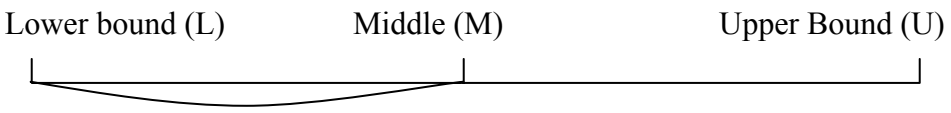
Two steps conduct the search process, which further determine the efficiency of a search process. The first step is to get a quality bound to make the search process fast. An appropriate lower bound can be found by computing objective functions through relaxed constraints. The second step is to determine a better solution with respect to a defined objective function or some other criteria. Computation in the second step is relatively costly. In fact, in many applications, users are satisfied with a solution that is close to optimum if this solution is found early. The B & B method can be used to find sub-optimal solutions using appropriate bound.

2.4.2 Dichotomic Search

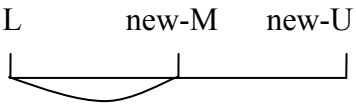
The dichotomic search improves the search efficiency over the standard search. This algorithm starts with a lower bound L and an upper bound U on the objective function z (x_1, x_2… x_n). The lower bound L is not necessarily a solution point, and it can be any arbitrary value reasonable for the problem. An initial solution to a CSP problem can serve as the upper bound U.

The dichotomic search procedure is essentially a binary search on the objective function. The midpoint M = (U+L)/2 of the two bounds is computed, and a CSP is solved by taking the design constraints and adding the constraint $z(x_1, x_2, \ldots, x_n) < M$. That means the search is in L ~ M region; see Fig. 2.4a. If a new feasible solution is found, then the upper bound is updated from M to the new feasible solution (new-U). A new middle point is calculated, named as new-M; see Fig. 2.4b. Then the search continues in area L ~ new-M as shown; see Fig. 2.4b. If no solution is found in L ~ M region the lower bound is updated to Middle point M. A new middle point new-M' is then calculated by (U+M)/2. The search continues within M ~ new-M' region (see Fig. 2.4c). This procedure will continue until the searching space is exhausted. The last lower bound will be taken as the optimum solution to the problem.

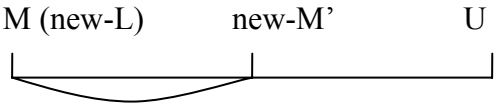
The dichotomic search is very effective when the lower bound is appropriate. The dichotomic search stresses on the search for feasible solutions, whereas the standard search emphasizes the improvement of the lower bound.



(a) Searching in L to M region (area indicated by curved line)



(b) Searching in L to new-M region (new-U: updated upper bound)



(c) Searching in M (new-L) to new-M' region.

Figure 2.4 Dichotomic Search

2.5 CSP Extensions

The CSP described above may be called the classical CSP or standard CSP. In some real-life applications, constraints may be incomplete at the time they are imposed. Therefore some alternative approaches to extend the classical CSP have been proposed for modeling incomplete constraints. Two of these extensions are described in the following paragraphs.

Fuzzy CSP: Fuzzy CSP extends the notion of the classical CSP by having constraints associated with a preference level to each tuple of values of variables. Such a level is represented by a number between 0 and 1, where 1 represents the best value (i.e., the tuple is allowed), and 0 the worst one (i.e., the tuple is not allowed). The solution of a fuzzy CSP is then defined as the set of tuples of values, which shifts preference level to a maximum.

Probabilistic CSP: Probabilistic CSP (Prob-CSP) enables to model those situations where each constraint c has a certain probability p(c), independent of the probability of other constraints, to be part of the given problem (actually, the probability is not of the constraint, but of the situation which corresponds to the constraint: saying that c has probability p means that the situation corresponding to c has probability p of occurring in the real-life problem). A solution to a Prob-CSP problem is found by satisfying that the expression Sum {Product {p(c) | c is a constraint in S} S is a subset of the set of all constraints satisfied by A} is maximized among all possible assignments A of values.

2.6 Preference Problems in CSP

Preferences in CSP refer to (1) preference on the value of the variable, (2) preference on the solution of a CSP problem, and (3) preference on the constraint. Examples of Preference (1) include: in the case of the elevator system, the door is preferred to open vertically; the color of the interior is white grey. Preference (1) can be specified in the CSP problem by

Door.open = 'Vertical'

Interior.color = 'White grey'

The left part in the above expressions represents the variables, and the right part represents the values.

Examples of Preference (1) include: in the case of the elevator system, the total cost of the elevator should be the lowest. Preference (2) can be specified in CSP as an optimization problem

(see the previous discussion in Section 2.4). Examples of Preference (3) include: in the case of the elevator system, if the elevator system is used in the hospital environment, the color of the interior is preferred to be light blue, and if in the general residence environment, the color of the interior is preferred to be grey.

Preference (1) and Preference (2) are, respectively, called Preference I and Preference II later in this thesis. Preference (3) is not considered to be in the scope of this thesis. Furthermore, it seems that not a sufficient research is done for Preference II, perhaps because it is classified into the general optimization problem. Yet, there is still some twist when the optimization in the context of CSP is implemented.

2.7 Limitations of CSP

There are basically two problems when a computational tool is concerned: (1) Its knowledge representation, and (2) its computational efficiency. The CSP representation provides a low level of the specification of relationships between two entities. These two entities are usually at the level of features of an artefact, although the syntax of CSP expressions does not exclude somewhat a heterogeneous representation, e.g., a feature of entity A is associated with entity B as a whole. The nature of the CSP representation hinders the efficiency of the solving process. The CSP approach needs heuristic rules to improve the efficiency, but this can hardly be achieved at the level of constraint specification. A novel solution is warrant and will be presented in the next chapter.

Chapter 3

An Integrated PDM and CSP Framework

3.1 Introduction

As analyzed in Chapter 1 and Chapter 2, respectively, there are several problems with CSP, especially when it is used directly for product life cycle in a virtual organization environment. This chapter will present solutions to these problems. Section 3.2 presents a general idea of the integration of PDM and CSP as one of the solutions to these problems. Section 3.3 presents a conceptual PDM for configuration design. Section 3.4 presents a conceptual model of CSP. Section 3.5 illustrates the linkage between the PDM and the CSP. Section 3.6 illustrates the ideas described in the preceding sections using the elevator system as an example. Section 3.7 concludes this chapter.

3.2 General Idea of Integration of PDM and CSP

The general idea behind this study is to integrate PDM and CSP. PDM has two roles in this connection. The first role is that PDM facilitates the engineering activities in the course of configuring a product; see Fig. 3.1a. Knowledge stored in the format of CSP is meaningful only in the sense of variables and constraints which are the relationships among the variables, while PDM promises to store all information and knowledge about a product over its life cycle [Krause

et al., 1993; Sinha, 2004]. For instance, the background information concerning why a particular part is interfacing with other parts, or why a particular colour is not available for the interior of an elevator will be stored in PDM but, certainly not always in CSP. In certain CSP systems, where the constraint may be represented by simply listing a pair of instances of two components, say A and B, e.g.,

$<a_1\ b_1>$ from the view point of colour match

$<a_2\ b_3>$ from the view point of size match

while the semantics of why these two instances are put together (i.e., the view point or context) are in the mind of the operator or administrator of that particular CSP package.

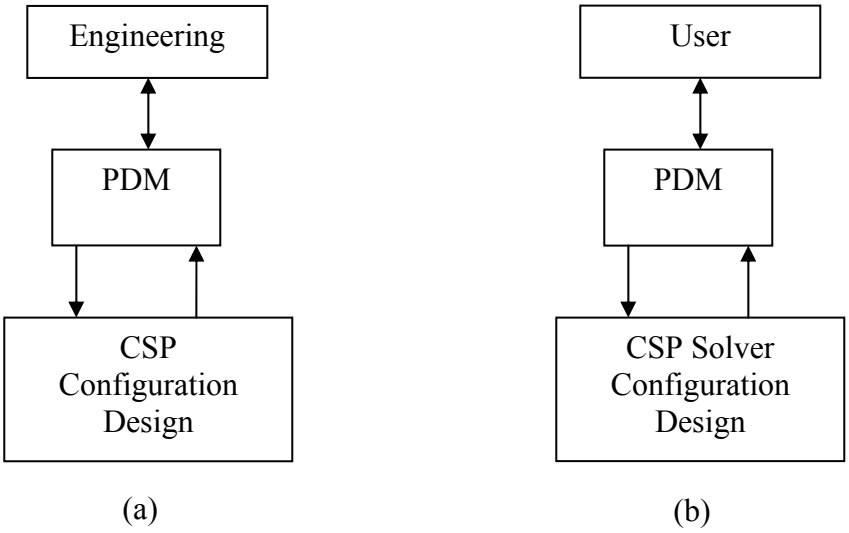


Figure 3.1 Integration Framework

The second role of PDM is that PDM provides semantic rich expressions such that it becomes a meta-knowledge representation for CSP; see Fig. 3.1b. For example, for a part, say a platform in the elevator system, there are two attributes, size and weight. The knowledge representation at the CSP level may go to the attribute level, which means that the operand in the CSP expression

may have the form (e.g., the platform), such as 'P01#.weight' and 'P03#.size,', where 'P01#' and 'P03#' are the component identities of the platform and the safety system, respectively, and 'weight' and 'size' are attributes or features of these components, respectively. At the CSP level, 'P01#' and 'P03#' do not make any sense because there is no expression clause that asserts any semantics for them. Furthermore, the fact that two pieces of information, i.e., 'P01#.colour' and 'P01#.size', are related to the same product, or represent two features of the same product, is missing at the CSP level. However, at the PDM level, there are representations to assert semantics to the instance 'P01#' (for example) through the schema – instance mechanism, and to assert the semantics that the weight and the size are two attributes of the platform.

Last, since PDM promises to contain complete information regarding a product under configuration, PDM can be a source of producing heuristic knowledge for improving efficiency in searching solutions to a CSP problem; see discussion in Chapter 2.

3.3 A Conceptual PDM for Configuration Design

As mentioned earlier, PDM captures all information about the product life cycle. There have been many PDMs proposed in literature [Krause et al., 1993; Shrikhande, 2000; Sinha, 2004]. The following is a set of core models with PDM:

- A data model for product assembly or architecture,
- A data model for product connectivity,
- A data model for requirement, and

- A data model for design knowledge base

The background information associated with these models is also captured and represented. In the following, these models are tailored to configuration design. In particular, Section 3.3.1 presents a data model for the requirement. Section 3.3.2 puts together the assembly and connectivity information (product configuration in short) and presents a data model for product configuration. Section 3.3.3 presents a data model for design knowledge for configuration design. These models are called conceptual because they do not depend on a particular implementation and they do not work for only a particular use.

3.3.1 A Data Model for Requirement

The design requirement includes: (1) the function, (2) the constraint, and (3) the wish. Examples of *functional* requirements are: the speed of the elevator must be greater than 300 feet/min; the capacity of the elevator must be 1000 lbs, etc. An example of a *constraint* requirement is: the color of the interior must be red. The constraint requirement may be converted into the functional requirement. The *wish* requirement includes the statements which represent the customer desire:

(1) Quality: as good as possible;

(2) Cost: as low as possible; and

(3) Time: as short as possible.

Note the wish may not necessarily be achieved, which differs from the function that has to be fulfilled, and from the constraint that has to be subjected to.

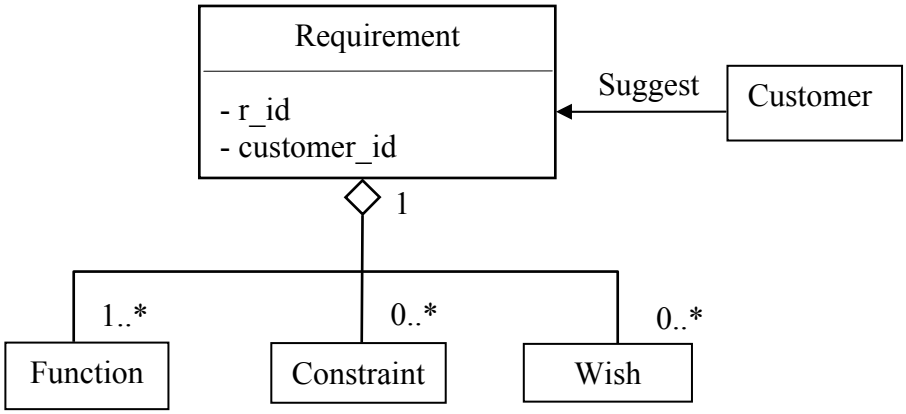


Figure 3.2 A Data Model for the Requirement

A data model for the requirement is shown in Fig. 3.2. Specifically, Fig. 3.2 represents the semantics that the design requirement is associated with the function, the constraint, and the wish. The multiplicity indicated on the diagram shows that there must be at least one function requirement; yet there may be none constraint or none wish requirement. It is noted that in Fig. 3.2, the customer who proposes the requirement is also shown. This is intended to track the requirement change initiated by the customer. It is the customer who keeps the rationale for a particular piece of requirement. So when the requirement is changed by a customer (say A), the customer (say B) who is associated with this requirement needs to be informed. For instance, a message is posted to customer B: "The requirement you suggested is to be changed by customer A." It should be noted that A and B may be the same person physically yet at different times. Here, it is shown that the proposed data model (i.e., Fig. 3.2) can also be useful for customer requirement management. Details about customer requirement management can be found in [Brown, 2000].

Fig. 3.3 represents the function requirement. There are two ways to specify the function requirement (see Fig. 3.3, F1 and F2, respectively). One way (F1) is, for example, the capacity of an elevator is 3000 lb, and the other way (F2) is, for example, the capacity of an elevator increases from 3000 lb to 5000 lb. In the data model for the function, the attribute 'product_feature' refers to both the structural and behavioural features of a particular product. For example, for the elevator system, the 'product_feature' takes the items (for example): 'capacity', 'platform.weight', 'platform.size', etc. The syntax 'platform.weight' has the following meaning. The first word 'platform' stands for a class, and the second attribute 'color' stands for an attribute of the class 'platform'. The generalization of this syntax is self-explanatory and applies to the remainder of the discussion in this thesis. The attributes 'operator', 'quantity', 'from_quantity', and 'to_quantity' are self-explanatory with respect to their corresponding classes (F1, F2). The two examples of the function requirement, as discussed before, can then be expressed in the form of instances as follows:

Instance of F1

<F001#, 'Capacity', '>', 3000>

Instance of F2

<F002#, 'Capacity', 3000, 5000>

Where the symbol '>' means 'greater than'.

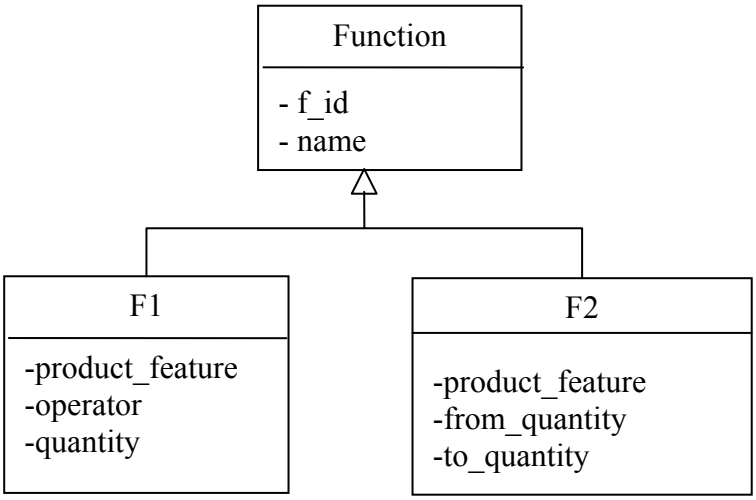


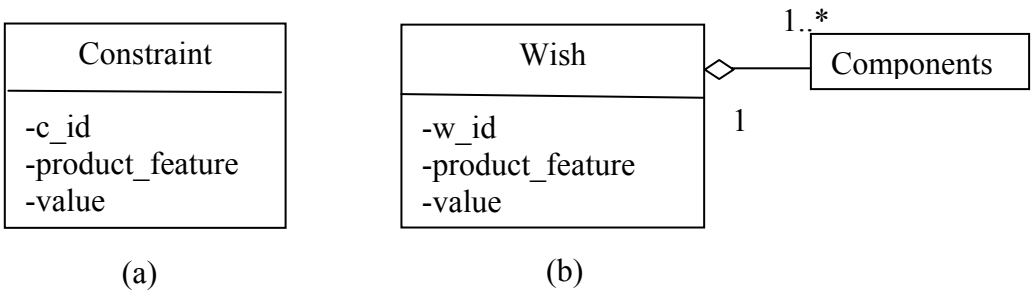
Figure 3.3 A Data Model for the Function

Fig. 3.4a is a data model for the constraint. The constraint is defined upon the structural feature of a product. The example of the constraint, regarding the color of the interior of the car in the elevator system, is:

Instance of constraint

<C001#, 'Interior.color', 'Red'>

where the term 'Interior.color' refers to the color of the interior of the elevator, and it is a structural feature of the elevator product.



..

Figure 3.4 Data Models for the Constraint and the Wish

Fig. 3.4b is a data model for the wish requirement. The wish may be applied to the whole product, or a set of components. An example of the wish requirement would be: the cost of an elevator system should be the lowest. The data representation of this wish requirement may be expressed in the form of instance as follows:

< W001#, Cost, Lowest, {component 1, component 2, …, component n} >

where the 'W001' stands for the identity of the wish; 'cost' is a behavioural feature of the product; 'lowest' represents the degree of the wish in the customer's mind. In the bracket '{… …}', all concerned components of a product are listed. When a whole product system (e.g., the elevator system) is described for a certain behaviour or property, simply put the name of the product in the brackets.

3.3.2 A Data Model for Product Configuration

The definition of product configuration is given in Chapter 1. In that definition, a product configuration is viewed as a network of connections among a set of components. Such a view of product is also called connectivity view. A product configuration may also be viewed as a set of

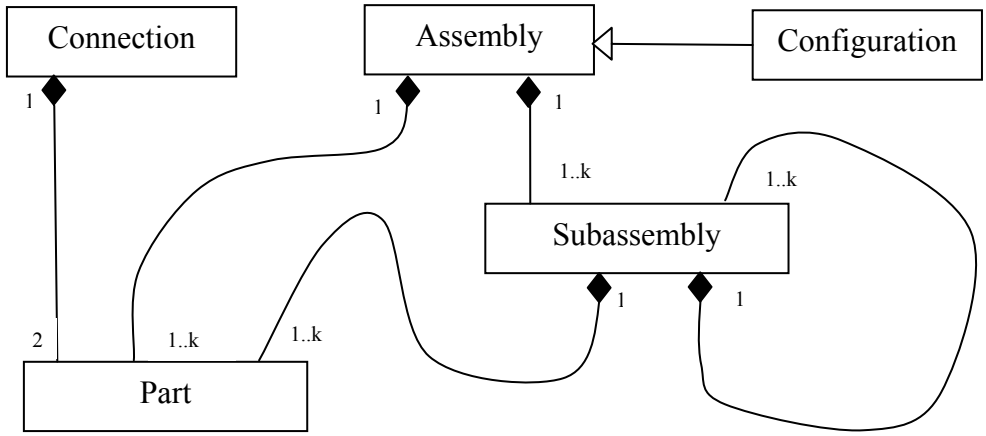


Figure 3.5 Data Model for the Assembly and Connectivity

sub-systems or components based on the function decomposition at varying levels. Such a view of product is called assembly view. For example, the elevator system is decomposed into (see Fig. 1.3):

- Hoist way assembly,
- Car assembly,
- C-weight,
- Suspension,
- Safety, and
- Cable.

The Hoist way assembly is further decomposed into the components: guide rail and U-bracket, and the car assembly are further decomposed into the subsystems: car and supporting system.

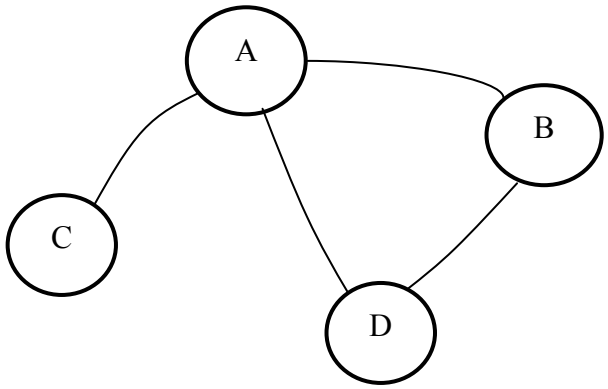


Figure 3.6 Pair Relationships for the General Connectivity Representation

A data model which captures the above semantics is shown in Fig. 3.5. In this figure, it is further remarked that the line which connects the class 'subassembly' to itself express that the level of decomposition of subsystems may be more than one and varying. Note that Fig. 3.5 also represents the connectivity view of product configuration. To make the model more general, the connectivity can be viewed as a set of pair relationships [Zhang and Van der werff, 1993]. For example, a product has four objects that are connected, as shown in Fig. 3.6. This connectivity view can be expressed by a list of pair relationships as follows:

<A, B>

<A, C>

<A, D>

<B, D>

Further, there are various types of connections in the product architecture, for example, the shaft-and-hole, face-to-face against, etc. Fig. 3.7 represents this semantics. Specifically, for the shaft-

hole connection type, a set of attributes that describe it is shown in Fig. 3.8, where the attributes of the class 'Shaft_Hole_Connection' are self-explanatory.

Furthermore, the data model illustrated in Fig. 3.5 also represents: a configuration is a kind of an assembly. In fact, the data model of assembly and connection is a generic one.

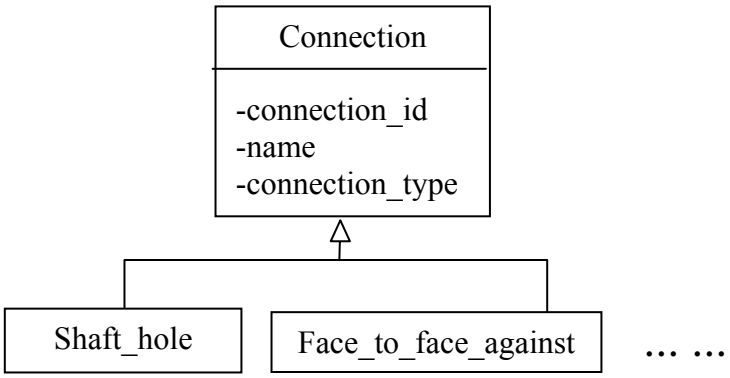


Figure 3.7 A Data Model for the Connection

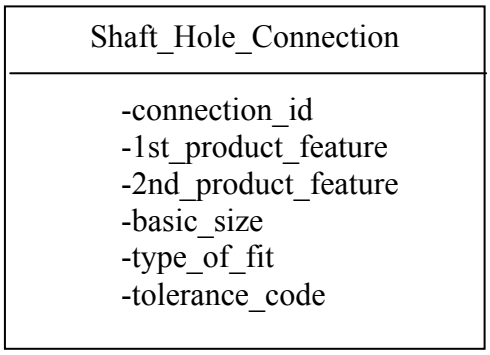


Figure 3.8 A Data Model for the Shaft-Hole Connection Type

There are linkages between the requirement data model and the product configuration data model. In particular, the domains of the attribute 'product_feature' should be included by the domain of the product ontology dictionary" (POD). The product ontology is the definition of

basic concepts and their relationships for a particular type of product, while the product ontology dictionary contains a list of such definitions. Therefore, the POD contains the following information:

- Structural features of components, of subsystems, and of systems,
- Behavioural features of components, of subsystems, and of systems, and
- Their semantics.

For example, in the POD, there may be the following definitions:

001#: Platform.size: the size of the platform;

002#: Interior.color: the color of the interior;

003#: Capacity: the capacity of the system;

004#: Cost: the effort which is converted into money for making products, subsystems, and components.

The first two examples are a structural feature of the product, while the last two are a behavioural feature of the product.

3.3.3 A Data Model for Configuration Design Knowledge

Design knowledge is mainly concerned with design synthesis knowledge; that is, it answers various “how-to-achieve” questions. The structure of design knowledge expression may depend on the general structure of a product. Generally speaking, the structure of a product has two

types: (1) serial type, and (2) parallel type. In the serial structure type, synthesis can be divided into n synthesis sub-tasks (see Fig. 3.9). In particular, synthesis at sub-task i is not subject to any constraint of that at sub-task j ($j \neq i$) except the input-output relation (i.e., output of sub-task i is input of sub-task i+1). In the parallel structure type, each component plays a unique role in forming a product; see Fig. 3.10.

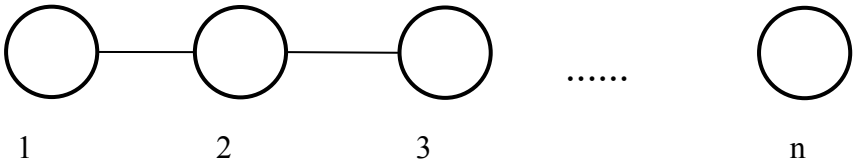


Figure 3.9 Serial Structure

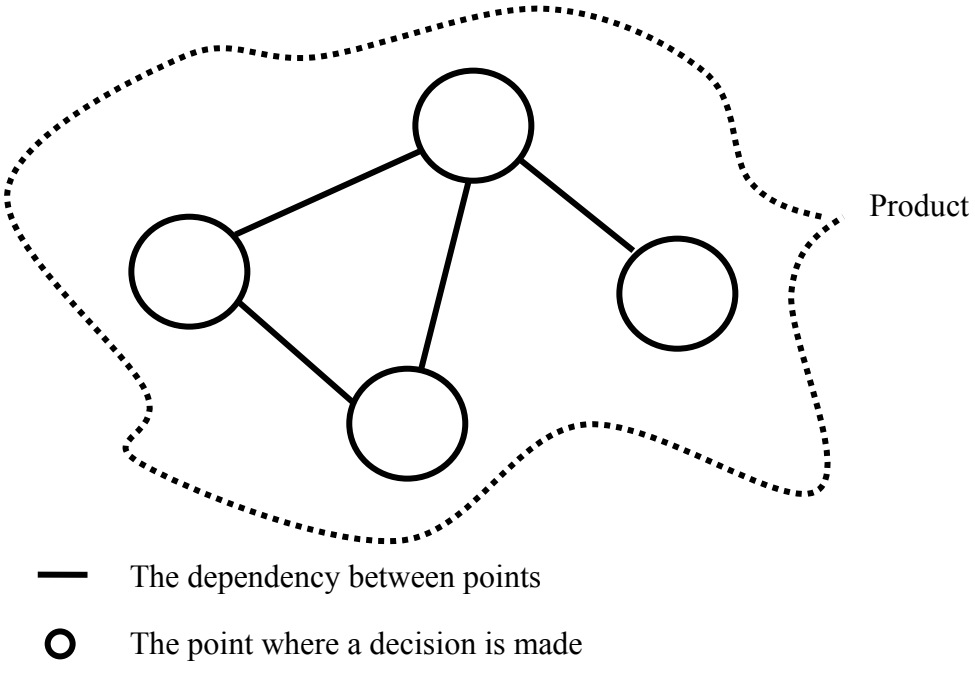


Figure 3.10 Parallel Structure

The representations of design knowledge for these two types of structures are considerably different. For the serial structure type, design knowledge can be expressed as a set of function-structure (F-S) pairs; see Fig. 3.11, where a sub-function further corresponds to one or more structures. The sense of "how-to-achieve" can be seen from the following scenario:

Given a functional requirement for a new product, the function is decomposed into a set of sub-functions, say F_1, F_2, …, F_n. A process is started to match the function-structure pair in the design knowledge database, which has the structure shown in Fig. 3.11. Assume that F_1, F_2, …, F_i ($i \leq n$) are matched. Then the function decomposition process will be continued on F_{i+1}, F_{i+2}, …, F_n. The design ends when all sub-functions are matched by structures, which means all functions are achieved. The design is thus a set of structures that can be obtained through the function-structure pair. For example, for F_1, one obtains S_1 (See Fig. 3.11). Note that this design process is well known, as described in design literature, e.g. [Suh, 1990].

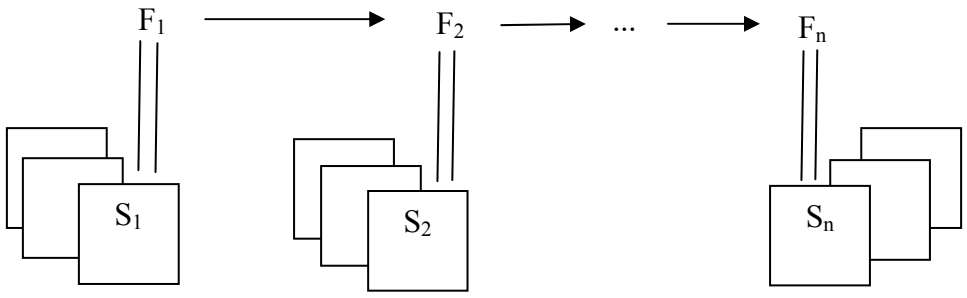


Figure 3.11 Serial Structure Design Process

For the parallel structure type, design knowledge is expressed in a way by determining all component instances "simultaneously" to satisfy a requirement. Design knowledge, in this case,

can hardly be expressed by separate blocks or units (the same way as for the case of the serial structure type). There may be several possibilities of combinations of component instances that achieve a requirement, and these possibilities are design knowledge for the parallel structure type. When the number of components, n, is large, the number of possible combinations of component instances is large. It is understood that the maximal number of possibilities is $m_1 \times m_2 \times \cdots \cdots \times m_n$ (m_i: the number of instances of component i) which can be a very large number. Instead of exposing all instances of components, the other method to express design knowledge for the parallel structure type is to describe "rules" that constrain the selection of instances of components. This method is more efficient, in the form of design knowledge expression, than the way that lists all combinations of component instances.

Configuration design is a design problem relevant to the parallel structure type design problem. Therefore, the serial structure type of design knowledge is not the concern in this thesis study. This thesis focuses on the parallel structure type of design knowledge. In the following, the data modelling issue for design knowledge for the parallel structure type is discussed.

Based on the above discussion, design knowledge for configuration design is in the form of various relationships and constraints. A constraint makes sense when a corresponding relationship must be maintained. For instance, the color of the interior of an elevator is related to the color of the door of the elevator; specifically the red of the interior matches the white of the door. When such a relationship needs to be maintained, a constraint is then built upon the color of the interior and the color of the door. Note that the relationship and the constraint are viewed interchangeably hereafter.

The following points of general constraints can be summarized. First, suppose there are two components/sub-systems, A and B. A has NA attributes (A_1, A_2, ……, A_{NA}), while B has NB attributes (B_1, B_2, ……, B_{NB}). Attribute A_i and attribute B_j may have a constraint, denoted by $<A_i, B_j>$. Second, the constraint may be applied to more than two components or attributes; for example, $<A_i, B_j, C_k>$, where A_i, B_j, and C_k are three components/subsystems, while the subscript i (j, k) indicates a specific feature associated with the component, or $<A_i, A_j, C_k>$, where two attributes of component A participate in a constraint. These types of constraints may be called the n-nary constraint (n>2). It should be noted that a n-nary constraint is not equivalent to two binary constraints, e.g.,

$<A_i, B_j>$ AND $<B_j, C_k>$

This is because these two binary constraints do not impose the constraint on A_i and C_k directly; A_i and C_k are constrained only in the sense that each of them is related to B_j. The following is an example to further illustrate this point. Consider that the constraints on three product features, A_i, B_j, and C_k, are, respectively (see Fig. 3.12),

$<a_{i1}, b_{j1}, c_{k3}>$, and

$<a_{i2}, b_{j1}, c_{k1}>$.

where the items in the symbol '<>' are the instances corresponding to A_i, B_j, C_k, respectively. For instance, a_{i1} is an instance of A_i.

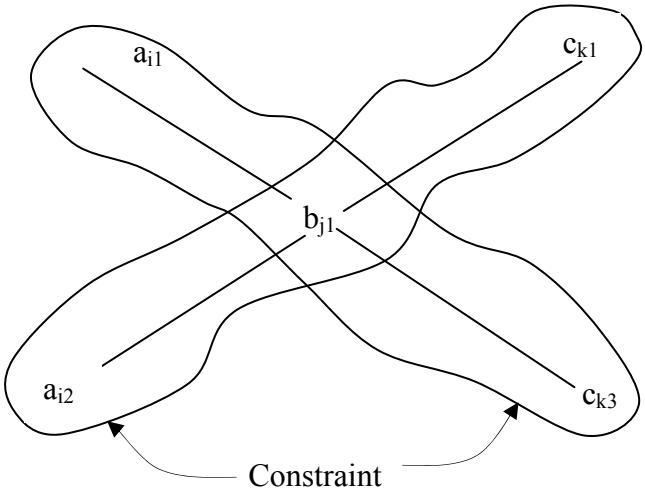


Figure 3.12 Constraints on Attributes

Now consider representing these constraints based on the binary constraints. This results in the following representation of instances:

<a_{i1}, b_{j1}>

<a_{i2}, b_{j1}>

<b_{j1}, c_{k1}>

<b_{j1}, c_{k3}>

The above representation is based on the idea that b_j is an intermediate which relates a_{i1} (a_{i2}) and c_{k1} (c_{k3}). However, such an idea is not able to exclude the following pieces of knowledge:

<a_{i1}, b_{j1}, c_{k1}>

<a_{i2}, b_{j1}, c_{k3}>

which are not part of the constraints for this problem. Therefore, the data representation of knowledge for configuration design should include the n-ary (n≥3) expressions.

Fig. 3.13 presents a data model of design knowledge for configuration design. It should be noted that the domain of the class 'structure_feature' and 'behavior_feature' are included by the domain of the class "POD" (see the previous discussion). Several remarks can be made regarding the data model shown in Fig. 3.13.

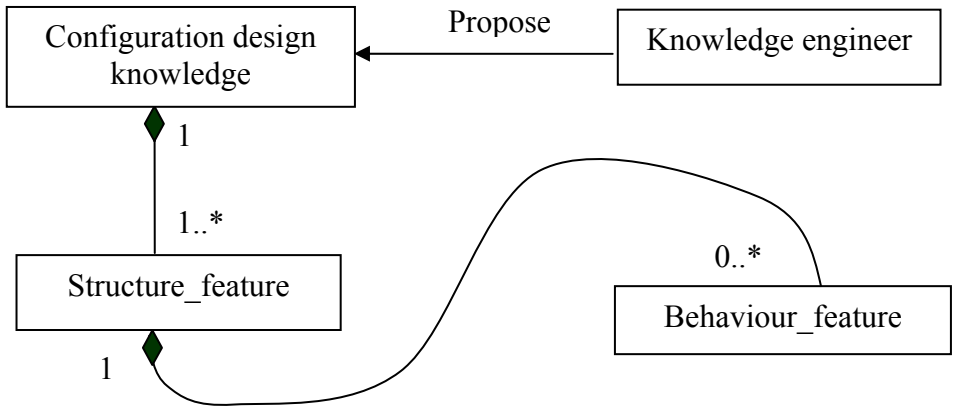


Figure 3.13 A Data Model for the Design Knowledge of Configuration Design

Remark 1: When the multiplicity indicted at the end of the class 'structure_feature' is '1', this means that there is only one 'structure_feature' associated with the class 'Configuration design knowledge'. In this case, the design knowledge which makes sense is simply the domain of the respective 'structure_feature'. The domain of an attribute ('structure_feature' in this case) serves as a kind of knowledge in the sense that any product feature outside the domain should not be considered as a valid design.

Remark 2: When the multiplicity indicated at the end of the class 'structure_feature' is '2', this means a binary constraint. When the multiplicity is '3', this means a ternary constraint, as discussed above, with special reference to Fig. 3.12.

Design knowledge is used by a designer for determining a configuration given a requirement. Fig. 3.14 captures this semantics. After a configuration design is finished, one obtains a pair of particular requirement instance and a particular configuration instance; see Fig.3.15. A set of such pairs forms a new knowledge base called the case-based configuration design knowledge base (CASE for short) (Fig. 3.16).

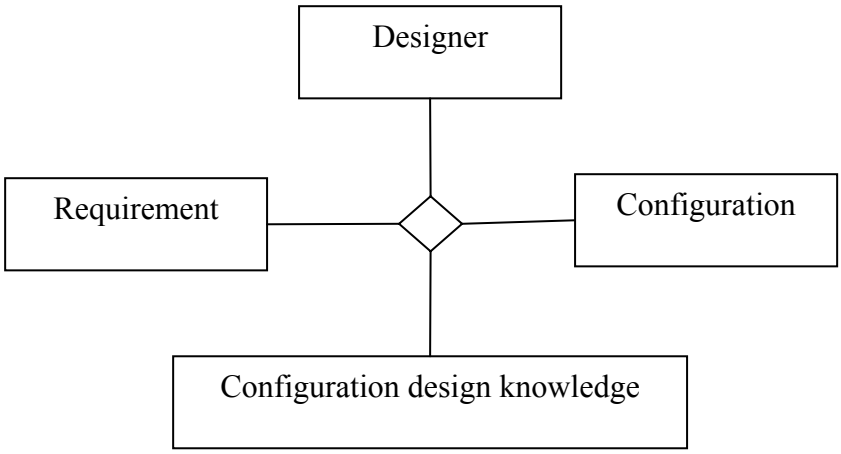


Figure 3.14 A Data Model for Configuration Design

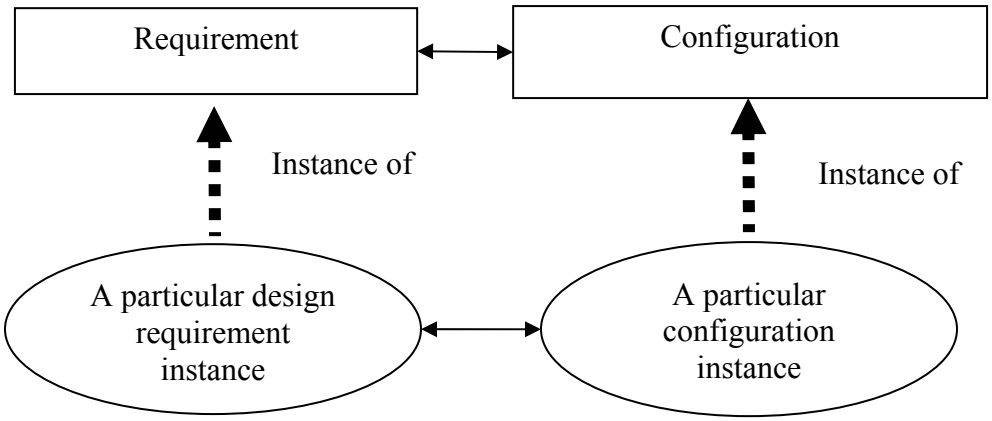


Figure 3.15 The Post Configuration Design

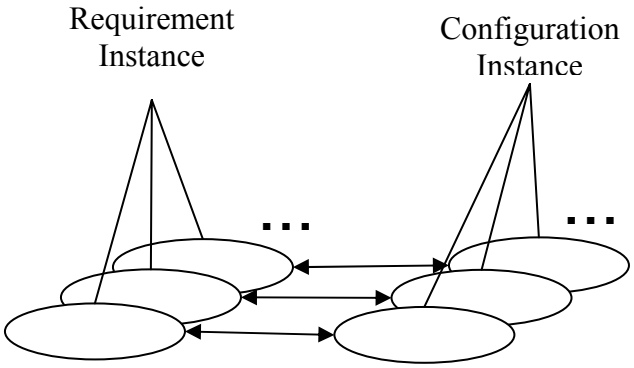


Figure 3.16 Case-Based Configuration Design Knowledge Base

CASE improves the efficiency of configuration design in the following sense. For any new configuration design task, the first step is now to search CASE by matching the requirement instance of a new design task. If there is a match, then simply retrieve the corresponding configuration instance. By this way one avoids searching configuration design knowledge base,

which may be a time-consuming task. On a general note, the idea described here may be called design process reuse.

3.4 A Conceptual Model of CSP for Configuration Design

The knowledge representation for configuration design, as proposed before, does not best suit computation. A computation-efficient representation usually requires such a formalism that includes the operand and the operator only. In the following, a conceptual model of CSP for configuration design is proposed. By the conceptual model it is meant that the representation is independent of any computer code which implements CSP. The conceptual model of CSP for configuration design is defined as follows:

Variable:

$$V_i \mid \text{Description}; \qquad i = 1, 2, \ldots, n \tag{3.1}$$

where n is the total number of variables, and i is the variable identity. It is noted that the domain can be viewed as the unary constraint.

Domain:

$$D_i \mid \text{A set of values} \tag{3.2}$$

Binary constraint:

$$C_j \mid \text{C-Expression} \tag{3.3}$$

where C_j is the constraint identity, and C-Expression takes the following form:

C-Expression: = operand 1 |operator |operand 2 (3.4)

Where operators include U, ∩, ≠, >, <, =, ┐, and "operand 1" and "operand 2" are variables defined in Equation (3.1). It is noted that these operations are not logic operators; they represent the semantics of various type of constraints.

In the case of the categorical type of variables, e.g., the variable 'color' which has the domain {'Red', 'Blue', …}, the operators that require quantitative information, such as >, =, ≠, <, become irrelevant. One needs to make use of the operators such as '∩', 'U', '┐'and ; see the discussion to follow. Regarding the (operators: U, ∩, ┐); specifically in the context of CSP, Variable 1 U Variable 2 means Either Variable 1 Or Variable 2; Variable 1 ∩ Variable 2 means Variable 1 AND Variable 2. Variable 1 ┐Variable 2 means Variable 1 incompatible with Variable 2.

In Chapter 2, two types of preferences, Preference I and Preference II, were discussed. Their CSP representations are as follows:

<u>Preference I:</u>

$P_i \mid PV_i \mid v_i \{a_i\}$ (3.5)

where

P_i : the preference identifier;

PV_i : the preference name;

v_i : the variable;

a_i : the value.

In the case of the elevator system, suppose that the customer prefers that the door is opened vertically. This preference can be represented as follows:

P002# | door opening | V_2 {'V_0'}

where

P002# : the preference identifier;

V_2 : the variable representing the door opening attribute;

V_0 : the value representing that the door is opened vertically

Preference I can be further represented as a form of the unary constraint. That is, for the door opening example, the specification would be

V_2 | {V_0}

Preference II:

P_i | PV_i | PII-Description | (3.6)

In Equation (3.6), PII-Description represents the preference of the customer on the whole system or sub-systems. In this study, the following types of Preference II are considered: (1) cost, (2) quality, and (3) weight. Therefore, there are three types of PII-Description.

PII-Description (1): Cost is the minimum | S_k

PII-Description (2): Quality is maximum | S_k

PII-Description (3): Weight is the minimum | S_k

S_k represents a system or a sub-system to which a preference is applied. For example, in the case of the elevator system, one may require the minimum weight of the sub-system consisting of the platform, sling, and crosshead. This preference can be specified as follows:

PII #01 | moving parts | PII #01 - Description |

PII #01 - Description = weight minimum | $\{V_1, V_2, V_3\}$|

V_1: Platform component

V_2: Sling

V_3: Crosshead

Preference II is solved by the CSP optimization technique, as mentioned in Chapter 2.

Composite constraint:

In the CSP literature, there is a notion called composite constraint specification [Sabin and Freuder, 1996]. Semantically, a variable (say VA) may represent a sub-system. The sub-system

consists of several components, which correspond to variables (say VB, VC, and VD). Thus, there is a hierarchical relation between VA and VB (VC, VD); see Fig. 3.17.

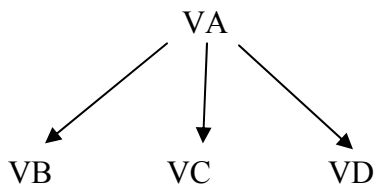


Figure 3.17 Composite Variable

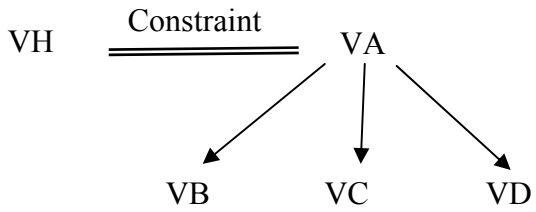


Figure 3.18 Composite Constraint

Suppose that there is another variable, say VH, which has a constraint with VA. This situation leads to Fig. 3.18. The specification of the composite constraint requires the possibility in CSP to define the hierarchical relation among variables.

3.5 From the PDM Model to the CSP Model

The representation of the CSP problem (in Section 3.4) can be completely derived from the knowledge representation for configuration design discussed in Section 3.3. In particular, the variable and its domain correspond to the attribute 'product_feature' (both the structural and behavioural features) and its domain in PDM. Preference I of CSP corresponds to the constraint requirement in PDM.

The function requirement defined in PDM level corresponds to the unary constraint in CSP. Here it should be noted that the semantics is gradually losing with the modeling process from PDM to CSP; see Fig. 3.19.

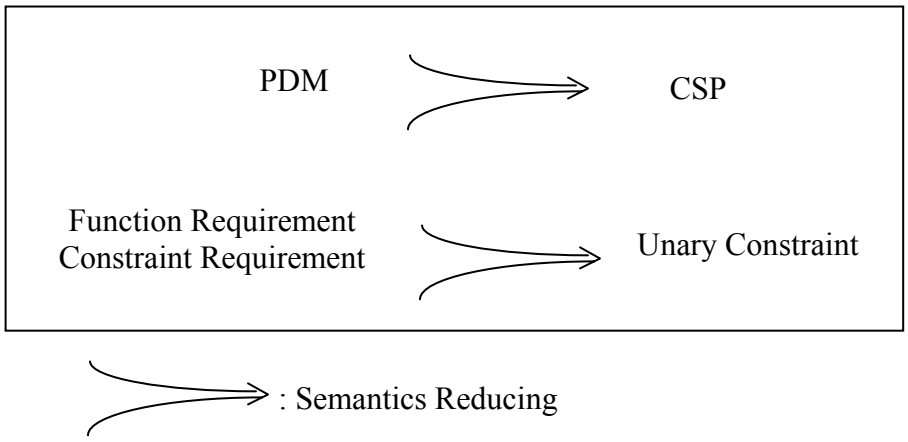


Figure 3.19 Modeling Process from PDM to CSP

The wish requirement in PDM corresponds to the Preference II in CSP. In PDM, a particular component may have to be associated with a particular subassembly. This semantics corresponds to the composite constraint in CSP; specifically, that partial configuration corresponds to a composite variable. It is noted that with the idea of the integrated PDM and CSP the relationship among variables for a composite variable is readily captured in PDM by the assembly and connection semantics (see Fig. 3.5).

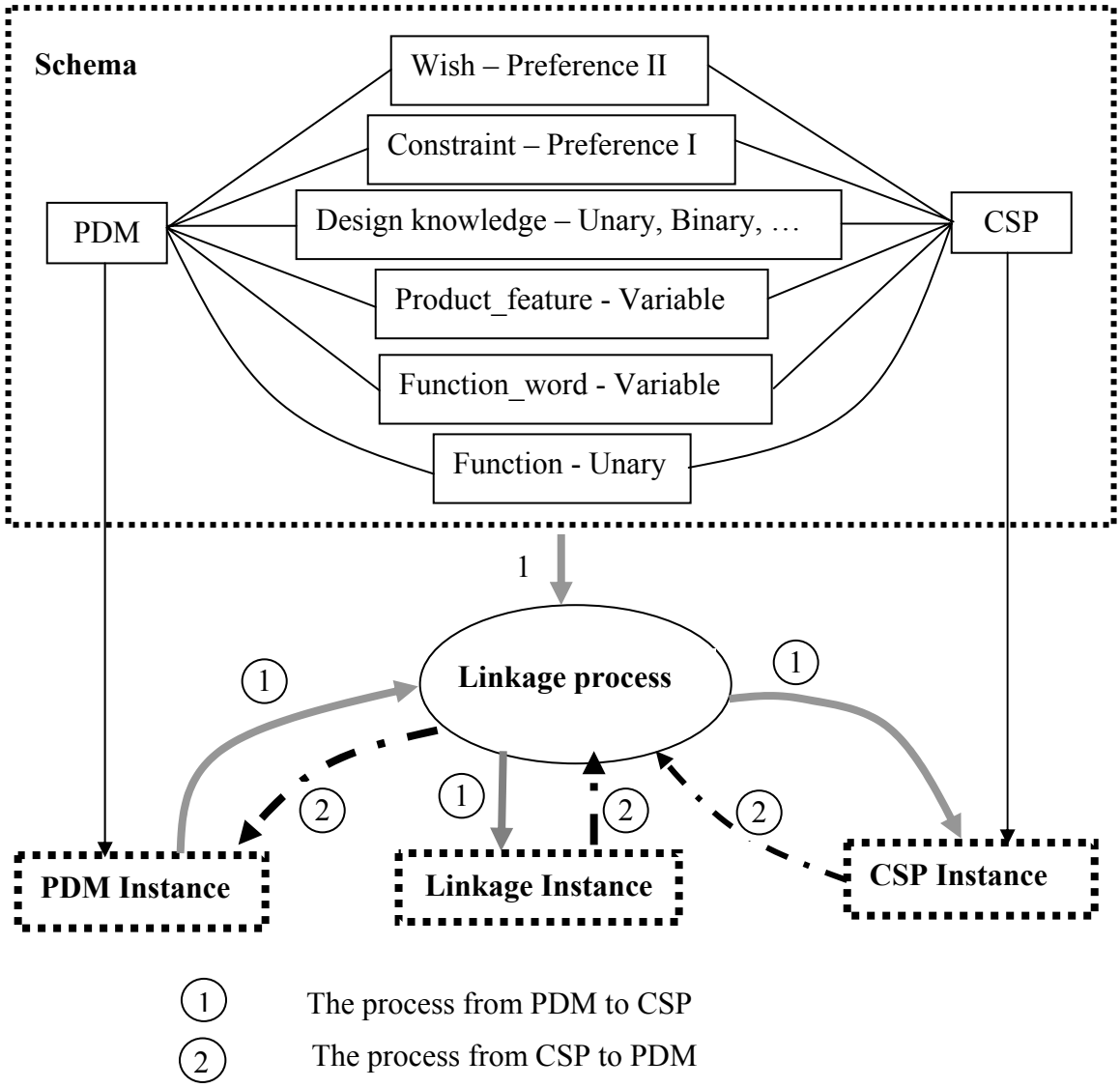


Figure 3.20 Linkage Data Model

Fig. 3.20 shows a data model which represents the linkage (discussed above) between the PDM and the CSP. The linkage data model makes it possible a two-way travel between PDM and CSP (also see Fig. 3.1b). A process, which may be called the linkage process, works upon the linkage model to maintain the two-way travel (see Fig. 3.20). In particular the linkage data model is automatically created when traveling from PDM to CSP, and after the computation for

configuration design is finished (at the CSP level) the process is back to the PDM level through the association between PDM and CSP in the linkage data model.

3.6 Illustration

The elevator system is used as an example to give an impression of what the models developed above look like. It is noted that the models developed before are “templates”, while a specific application problem is “instances” of these “templates”.

3.6.1 Configuration Design Requirement

The elevator system, as earlier mentioned in Chapter 1, consists of 16 components and 4 optional components. Each component has a certain number of features or attributes, and they are given below.

Door:	Features:	(1) open type
		(2) speed
		(3) open strike side
Platform:	Features:	(1) width
		(2) depth
Sling:	Features:	(1) size
		(2) type
Safety	Features:	(1) size
		(2) type

Head	Features:	(1) size
Carbuffer	Features:	(1) size
		(2) type
Cweight	Features:	(1) size
		(2) type
Sheave	Features:	(1) capacity
Cwtbuffer	Features:	(1) size
Concable	Features:	(1) type
Hcable	Features:	(1) type
		(2) duty
Gcable	Features:	(1) type
Comcable	Features:	(1) type
Machine	Features:	(1) capacity
Mbeam	Features:	(1) size
Motor	Features:	(1) power

The design requirement of the elevator system may include the following specifications (not a complete list):

Function:

(1) Elevator capacity: 2500 lb

(2) Elevator speed: 250 ft/min

(3) Need a phone or not: yes

(4) Need a lantern or not: no

(5) Need a communication or not: no

(6) Need an indicator or not: no

Constraint:

(1) Door open type: side open

(2) Door open speed: double

Wish:

(1) The lowest cost of the whole elevator system

3.6.2 The PDM representation of Design Requirement

The PDM for the above design requirement is presented below:

Following the data model shown in Fig. 3.3, the instances of the data model representing the semantics of the function requirement are shown below:

<F001#, ‘Capacity’, ‘>’, 2500 lb>

<F002#, ‘Speed’, ‘=’, 250 ft/min>

<F003#, ‘Phone’, ‘=’, ‘yes’>

<F004#, ‘Lantern’, ‘=’, ‘no’>

<F005#, ‘Communication’, ‘=’, ‘no’>

<F006#, ‘Indicator’, ‘=’, ‘no’>

Following the data model shown in Fig. 3.4, the instances representing the semantics of the constraint and wish requirements, respectively, are shown below:

Constraint:

<C001#, Door.open, “Side open”>

<C002#, Door.speed, “Single speed”>

Wish:

<W001#, ‘Cost’, ‘lowest’, { }>

where ‘{}’ implies that this particular wish is applied to the whole system (i.e., the elevator).

3.6.3 The CSP Representation of Design Requirement

Variables (the left side are the variable names, and the right side is the description)

V_0 | Door

V_1 | Platform

V_2 | Sling

V_3 | Safe

V_4 | Head

V_6	Car buffer
V_7	Car weight
V_8	Sheave
V_9	Cwtbuffer
V_{10}	Concable
V_{11}	Hcable
V_{12}	Gcable
V_{13}	Comcable
V_{14}	Machine
V_{15}	Mbeam
V_{16}	Motor
V_{18}	CarPhone
V_{19}	CarLantern
V_{20}	CarIntercom
V_{21}	CarIndicator
V_{22}	ElevatorCapacity
V_{23}	ElevatorSpeed

There are other variables, which refer to the design requirement, and they are presented as follows:

VF01	required capacity
VF02	required speed
VF03	required door type and speed

VF04	need a phone or not
VF05	need a lantern or not
VF06	need a communication system or not
VF07	need an indicator or not
VFCOST	cost of the total system

In the above,

VF01 corresponds to V_{22}

VF02 corresponds to V_{23}

VF03 corresponds to V_{0}

VF04 corresponds to V_{18}

VF05 corresponds to V_{19}

VF06 corresponds to V_{20}

VF07 corresponds to V_{21}

The design requirement is represented in CSP as follows:

Function:

VF01 = 2500 lb

VF02 = 250 ft/min

VF04 = ‘yes’

VF05 = ‘no’

VF06 = ‘no’

VF07 = ‘no’

Constraint:

The design constraints (1) and (2), as described above in Section 3.6.1, are represented by one variable in CSP.

VF03 = ‘2sso’

Where ‘2ssso’ stands for (1) the door is ‘side open’, and (2) door open speed is ‘double’

Wish:

VFCOST = ‘lowest’

3.6.4 Design Knowledge Representation

Design knowledge stated at the application level:

For example, Car Assembly made of **Door**, **Platform**, **Sling** and **Crosshead** which are called components. Each component has several models which fulfill similar function but each of them meets different requirements from geometric, physical, structural aspects. Take the component Door and Platform as an example. Their alternative models (in parentheses) are listed as follows:

Door: (ssco, ssso, 2sco, 2sso)

where ssco : single speed and center open;

Ssso : single speed and side open;

2sco : double speed and center open;

2sso : double speed and side open.

Platform: (2.5B, 4B, 6B)

where 2.5B : the smallest model;

6B : the largest.

The **platform** model (2.5B) is compatible with the **door** models (ssco, ssso) but not the door models (2sco, 2sso). Further, there is design knowledge which says: If **Capacity** is equal to or less than 2500 lb, use **Platform** model **2.5B**. If **Capacity** is larger than 2500 lb and less than 3000 lb, use platform model **4B**; otherwise use platform **6B** model.

Design knowledge represented in PDM:

In the production configuration data model, one has the following instance:

Assembly:

<A01#, P01#, P02#, P03#, P04#>

Part:

<P01#, 'Door'>

<P02#, 'Platform', 'Capacity'>

<P03#, 'Sling'>

<P04#, 'Crosshead'>

For representing the feature values, a general data model proposed by Zhang and Van der Werff [1993] is applied. This data model suggested two attributes in a class: the attribute id and its value. Therefore, one has the following definition of the structure feature.

Structure_feature:

<PF001#, P01, ssco>

<PF002#, P01, ssso>

<PF003#, P01, 2sco>

<PF004#, P01, 2sso>

<PF005#, P02, 2.5B, {#, 2500}>

<PF006#, P02, 4B, {2500, 3000}>

<PF007#, P02, 6B, {3000, #}>

Configuration design knowledge:

<DK001#, PF005#, PF001#>

<DK002#, PF005#, PF002#>

Design knowledge represented in CSP:

In the CSP level, the design knowledge is represented by

The Domain:

Door: {ssco, ssso, 2sco, 2sso}

Platform: {2.5B, 4B, 6B}

Capacity: {2000, 2500, 3000, 3500, 4000}

The Binary relation:

#089 | C- platform.2.5B ∩ door.ssco

#090 | C- platform.2.5B ∩ door.ssso

#091 | C- Capacity.C1 ∩ platform.2.5B

#092 | C- Capacity.C2 ∩ platform.4B

#093 | C- Capacity.C2 ∩ platform.6B

In the above, C1, C2, and C3 are so-called surrogates which are defined as follows:

C1: capacity less than 2500 lb;

C2: capacity less than 3000 but greater than 2500;

C2: capacity greater than 3000.

The linkage between PDM and CSP is illustrated as follows:

PDM		**CSP**
P01	corresponds to	Door
P02	corresponds to	Platform
Structure_feature	corresponds to	Domain of door and platform
Configuration design knowledge	corresponds to	Binary constraint
Behaviour_feature	corresponds to	Domain of Capacity

3.7 Concluding Remark

The integration of PDM and CSP is a promising idea for a more intelligent computer system for configuration design. The improved intelligence is because this idea gets both the strengths of the two paradigms (PDM, CSP): the rich expression of application semantics with PDM and the powerful facility (in computation) and generality (in knowledge representation) with CSP. The essence of this idea is such that PDM becomes a "wrapper" over CSP; CSP is merely a computational engine. One can also see that at the CSP level, the binary constraint has difficulty in representing the continuous variables (i.e., capacity <2500). At the current CSP formalism, the notion of surrogate was applied.

Chapter 4
Life Cycle Configuration Design

4.1 Introduction

A configuration design system or configurator does not work in isolation from other design and manufacturing processes. The life cycle configuration design concept will be proposed in this chapter. Specifically, Section 4.2 will elaborate this concept. Section 4.3 presents a general framework for integrating various life cycle systems and two specific integrations, namely the Configurator with CAD (Computer Aided Design) and the Configurator with SCM (Supply Chain Management). Section 4.4 is a summary.

4.2 Life Cycle Configuration Design Concept

The product production is a process which takes a triple (energy, material, information) in and generates a new triple (energy, material, information) out; see Fig. 4.1. Such a process is a controlled process with some goals to achieve and subject to some constraints; see Fig. 4.1. The things to be controlled or managed are resources, such as equipment, people, money, subcontractors, and suppliers; see Fig. 4.2. In literature, control of equipment and material may be fulfilled by a software system called “Manufacture Resource Planning (MRP)”. The control of financial flow may be fulfilled by a software system called “Enterprise Resource Planning (ERP)”. Control of supplies may be fulfilled by a software system called “Supply Chain

Management (SCM)", and finally management of customers may be fulfilled by a software system called "Customer Relation Management (CRM)". These software systems may not stand-alone; they may physically be integrated into one software system. For instance, many ERP systems include the functions of CRM and MRP.

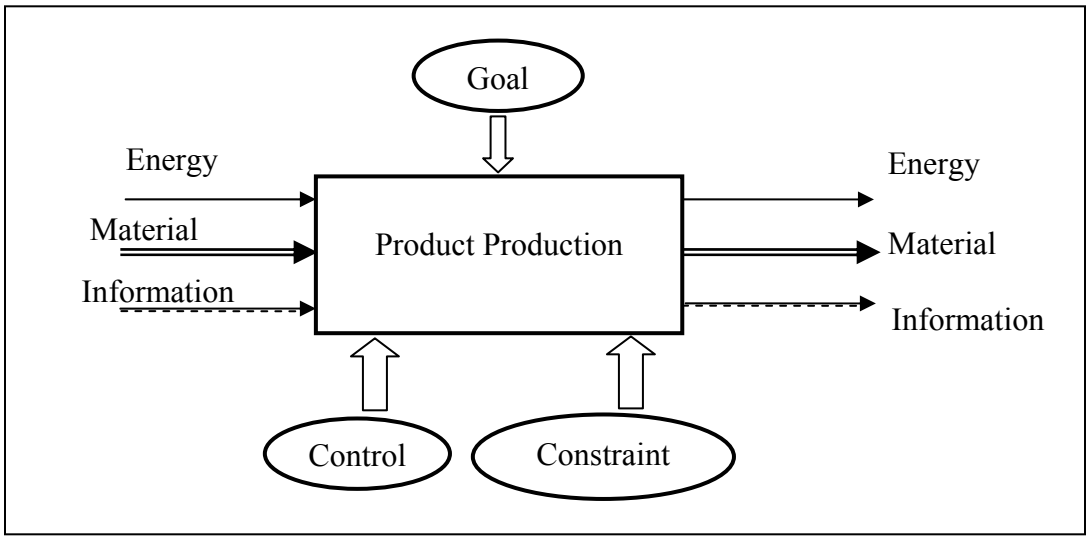


Figure 4.1 General Product Production Process

The concept of life cycle configuration design (LCCD) means that the configuration design activity involves all these management or control activities. It should be noted that product life cycle design or engineering for general products was first elaborated by Alting and Legarth [1995]. Here, configuration design is viewed as a particular design pattern with respect to general product design. Therefore, the life cycle (general) idea should be well applied to configuration design. In order to develop an integrated life cycle configuration design system, the idea proposed in this thesis is to identify interfaces between the configurator and many other control/management systems.

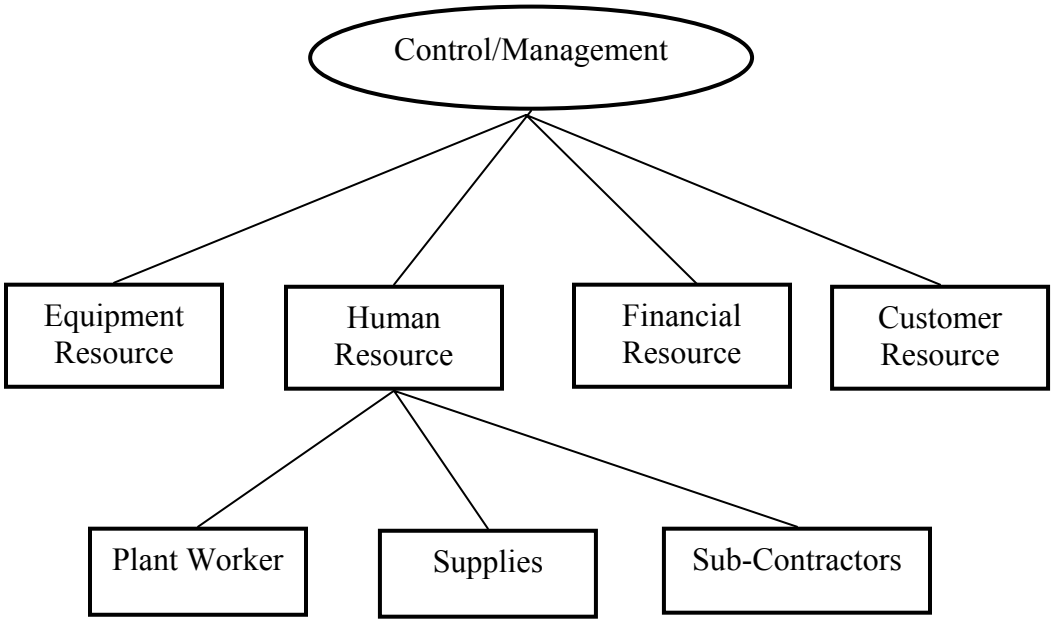


Figure 4.2 Things to Be Managed

4.3 Interfaces between Configurator and Other Life Cycle Programs

With the concept of integrated PDM and CSP (hereafter, PDM-CSP for short), the interface between configurator and the other systems as mentioned before (e.g., CAD/CAM system, ERP system) simply become the interface between PDM and the other systems. In general product production, interface between PDM and the other systems is well known, specifically under the heading called Product Life Cycle Management (PLM). In fact, the early system with the heading of PDM has already taken the whole life cycle of product into consideration; see Shrikhande [2000]. Nevertheless, there are two reasons that a study of PLM or PDM issue with special reference to configuration design is warrant here. *First*, the configuration design has its special features that may demand special considerations. *Second*, the current development of

XML allows the possibility to develop a mixed structural and semi-structural data that was recognized in [Shrikhande, 2000] as being very important for an effective PDM.

This thesis was not intended to give a full model for the life-cycle configuration design system but to focus on general framework and concept. The following discussion presents the concept of interface as a key technology for gluing a configurator with other program systems that support life cycle configuration design and illustrates how this concept works based on a few examples.

4.3.1 The Interface Framework

Fig. 4.3 illustrates a general framework of interface between a configurator with other program systems. In this framework the configurator and those other programs have a port which stores information and knowledge that is ready to communicate with other programs. Between the ports is an interface system. The interface is basically a kit of tools that further facilitate the communication. Here, the communication is mostly about data flow in the sense that the data from the configurator flows to the program (e.g., a CAD program), or vice versa. It is assumed that program A will not directly intervene program B in such a way program A, serving like a process of program B, changes data in the data repository of program B. In other words, each program has a process which represents intelligence of that program; see Fig. 4.4. Such a process is refreshable and replaceable, the same as human learning.

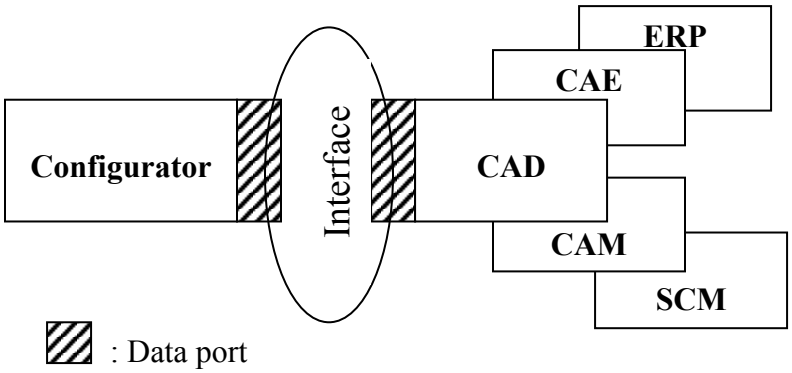


Figure 4.3 The Interface Framework

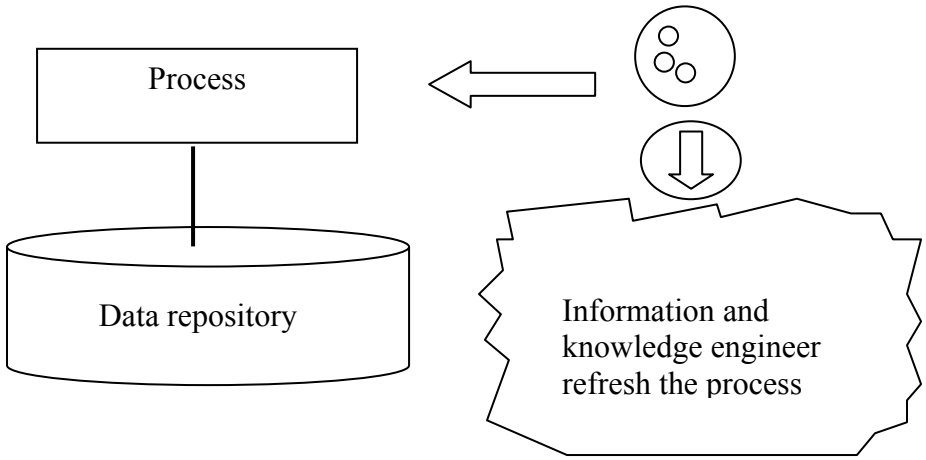


Figure 4.4 The Concept of The Process of a Program

It is further noted that the data in the port of a program (e.g., configurator) may differ depending on what other programs it communicates with. For example, when a configurator communicates with an ERP program, the data from the configurator's port may only include the so-called bill-of-material (BOM) data, which is a list of materials, their quality, and their quantities. Note that BOM does not contain the structure over these materials (i.e., a kind of assembly of these materials). The data in the configurator's port may contain the product assembly and

connectivity information, as represented with the model illustrated in Fig.3.5 (Chapter 3), when it communicates with a CAD program system.

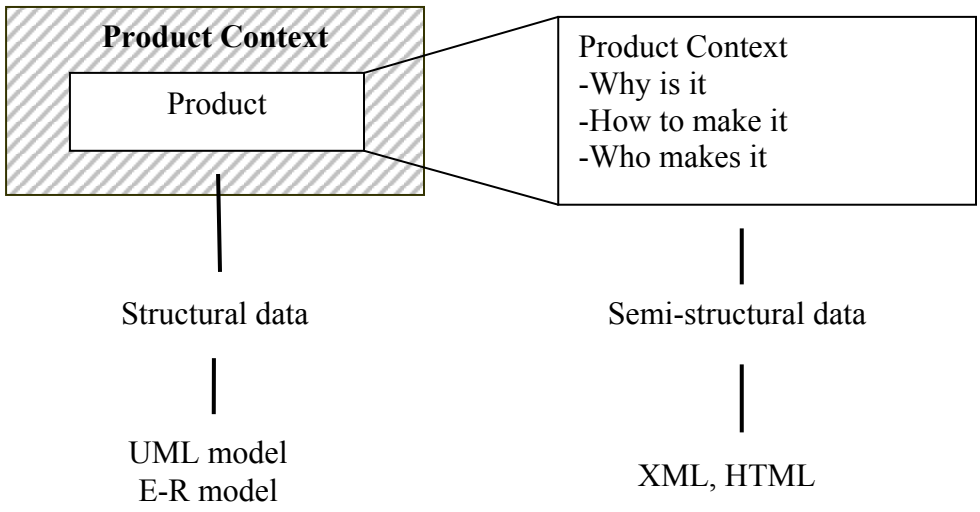


Figure 4.5 Structural data vs. Semi-Structural Data

Data in a program's port typically contains both structured data and semi-structured data. The structured data is one that conforms to a particular data model framework (e.g., relational, etc.), while the semi-structured data does not. The semi-structured data usually provides the context data for the structured data; see Fig. 4.5.

Recently, XML emerges to be a powerful tool for representing semi-structured data [Marchal, 2000] (more details about XML refer to Appendix A). The structured data and semi-structured data must be integrated, as the latter is the context of the former (Fig. 4.5). The key to make the integration possible is that they both share the same ontology; see Fig. 4.6.

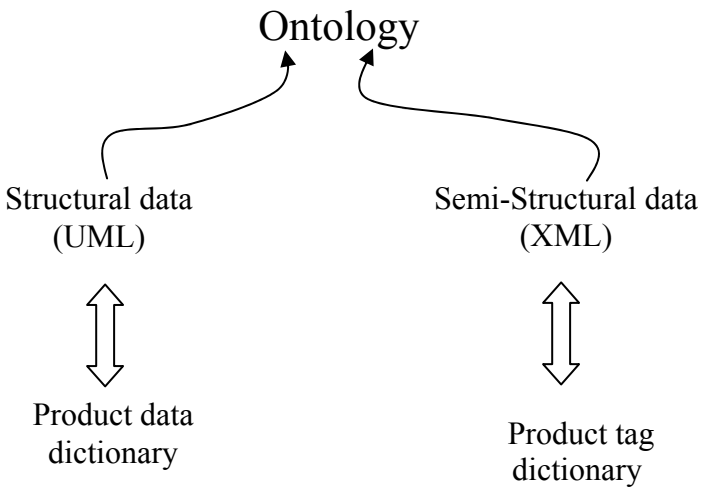


Figure 4.6 Integration of Structural and Semi-Structural Data

4.3.2 The Interface between Configurator and CAD Program

(1) Data in the configurator's port:

Structural data:

- Product assembly and connection model (see Fig. 3.5), and
- Parts Catalogue.

Semi-Structural data:

- Customer information,
- Configuration design identifier,
- Design id, and
- Design status (design, review, or release, etc.).

(2) Data in a CAD system (e.g., SolidWorks):

- Assembly and parts in CAD data format (e.g., DWG, IGES, STEP, ACIS, DXF, etc.),
- Mates relationship, and
- Title block information (designed by, checked by, approved by, etc.).

(3) Interface:

The data transfer of the structural data from the configuration's port to the CAD program's port may be performed by a human operator if there is a lack of the automated transfer process. Take the elevator as an example. This means the personnel who administrate the configurator needs to create a solid model of a configured product in a particular CAD system, e.g., SolidWorks. It is noted that such a model should be parametric for the configuration design application. Once the CAD model is established, the subsequent design scenarios will be as this. After a solution is generated by a configurator, the configurator will send parametric information to the CAD system by specifying a particular product family (e.g. elevator) stored in the CAD system. Then, the CAD system will realize the solution (i.e., a configured product) to the customer. The customer may be allowed to do change directly on the solution in the CAD system if the customer is familiar with the CAD system (e.g., SolidWorks) or if a user interface system, which relieves the customer of the need to be familiar with the particular CAD system, is available.

There is also a data transfer of the semi-structural data (XML format) from the configurator's port to the CAD's port. For example, the XML representation of the semi-structural data is shown in Fig. 4.7. It is noted that in this figure the tag words used in the XML representation are

drawn from the product ontology dictionary (POD) as discussed in Section 3.5. In that sense, the structural and semi-structural data are integrated.

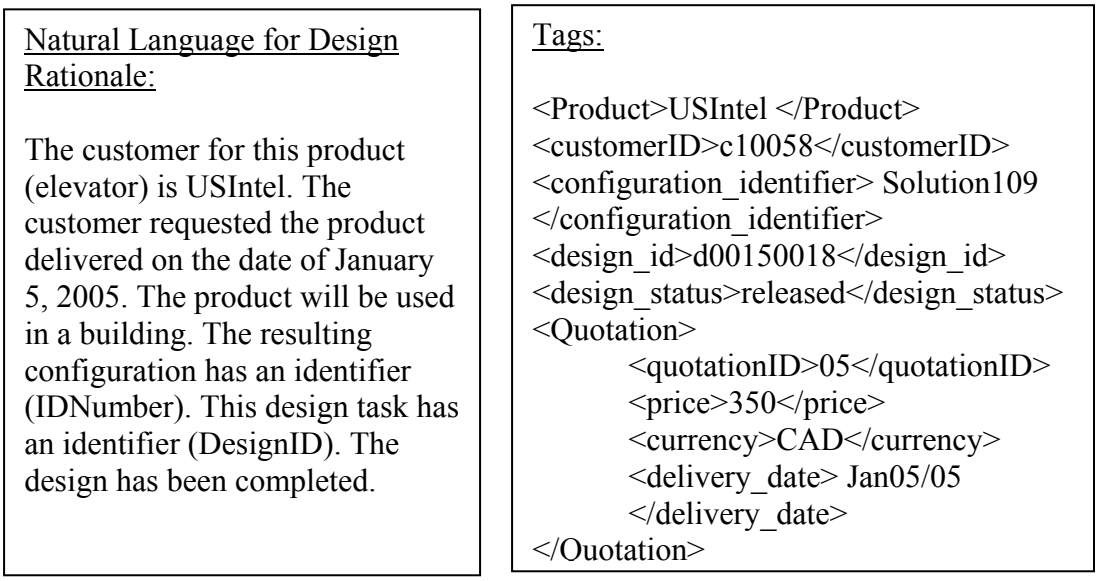


Figure 4.7 XML Representation of The Semi-Structural data

4.3.3 The Interface between Configuator and SCM

SCM describes the integration beyond boundary of the firm, involving business partners in different processes and activities. It involves suppliers, producers, retailers, logistics-service providers, and customers. By making planning cooperatively by all the participants it is believed to be more cost effective than isolated planning by individual participant. Therefore the communication for information exchange is very important for a seamless integration over all players, customers and its partners.

Electronic Data Interchange (EDI) system has been one widely used to achieve integration over participants. However EDI's considerable setup time and high operation costs stop many small

and medium size companies from using it. XML, instead, is very promising for representing structured and semi-structured data in SCM with low cost or even 0 cost [Buxmann et al., 2002].

The following is an example of data for the communication between a Configurator and a SCM program. Suppose that the end-user (customer who needs a product) is satisfied with the solution (a configured product) generated by the Configurator. The Configurator will need to communicate with the SCM system in order to generate the delivery date for the customer. For this purpose, at the Configurator's port, the BOM data is to be sent to the SCM system. Some background information may be attached with the BOM data. The example of such background information is the customer required delivery date and the customer location. The SCM system receives the Configurator's request and will then analyze the request and develop a supply plan for which the SCM system may further communicate with potential supplies. After the SCM system has worked out a supply plan, the SCM system will then return the result to the Configurator, and the Configurator generates the product delivery date to the customer and in the mean time the customer will be given the access to the SCM system to track the product delivery process. The interface between the Configurator and the SCM in this case will not have much difficulty in terms of the data format; the text format for the BOM and XML format for the other information should suffice. For example, the XML file for the background information (as mentioned before) can be formed in Fig. 4.8.

```
<Background>
   <CustomerLocation>
      <StreetNo>210 ABC Ave. </StreetNo>
      <City> St. Peterburger</City>
      <Province>Saskatchewan</Province>
      <PostalCode>S7H1A7</PostalCode>
   </CustomerLocation>
  <DeliveryDate> Jan20/05</DeliveryDate>
 </Background>
```

Figure 4.8 Background Information of a Part in XML

4.4 Summary

An effective configuration design system will have to provide a facility for life cycle considerations. The essential issue here is integration of a configurator with various other life cycle design systems, e.g., CAD system, SCM system, ERP system, etc. The general solution is to view the problem as an interface problem. In this chapter, the integration of the configurator with two life cycle modules, CAD and SCM, was discussed at the conceptual level (i.e., with emphasis on what information and knowledge are required in order to make integration happen). It has been long recognized that information and knowledge has to address both structural and semi-structural data. These two categories of data represent, respectively, decisions and their rationale. An idea was described that is to have UML for structural data and XML for semi-structural data, and further let two data models (UML and XML) share the same ontology of a domain of application under consideration.

Chapter 5
Implementation and Demonstration

5.1 Introduction

This chapter discusses an implementation of the theoretical developments described in the proceeding chapters. The purpose of this implementation is to verify whether the concepts proposed in the preceding chapters can be implemented both effectively and efficiently. In this case, the elevator system is taken as an example. The implementation resulted in a web-based integrated Configurator software system for the elevator system development. The configurator was based on the integration of PDM and CSP (an idea discussed in Chapter 3); specifically PDM wraps CSP. Hereafter, this system is called the PDM-CSP Configurator (or Configurator for short). Section 5.2 presents the system architecture of the PDM-CSP Configurator. Section 5.3 discusses the general implementation methodology. Section 5.4 discusses an implementation of CSP. Section 5.5 discusses an implementation of the integration of the Configurator and the CAD system. Section 5.6 presents a demonstration.

5.2 The architecture of PDM-CSP Configurator

Architecture of a system describes the various software modules and their relationships. It is well known that the architecture of a complex software system should make the following three components separate: the presentation, the data, and the process [Zhang, 1994]. This idea is translated to the web-based software application, which results in the structure shown in Fig. 5.1.

In this figure it can be seen that there is a server layer which refers to the CSP algorithm and the knowledge conversion program between PDM and CSP, an interface layer which refers to the interaction of the server with the client or customer, and a database layer which stores information and knowledge needed for conducting configuration design. It should be noted that with this architecture, data management is performed at the server layer in a centrally controlled manner. The client can start with PDM-CSP at any location. At the client end, a standard browser is needed, such as Netscape or Internet Explorer.

Each layer contains several program modules. At the database layer, there are (1) information about the design of a particular product, (2) the knowledge for designing the product, and (3) some general product development knowledge (e.g., materials, supplies, etc.). At the sever layer, there are (1) the database management system (DBMS), (2) the data conversion system (i.e., the linkage process; see the discussion in Section 3.5), and (3) the main program to perform configuration design. At the interface layer, there are (1) the interface management system which deals with the interface structure, (2) the user-model, and (3) some conversion program (e.g., XML to HTML).

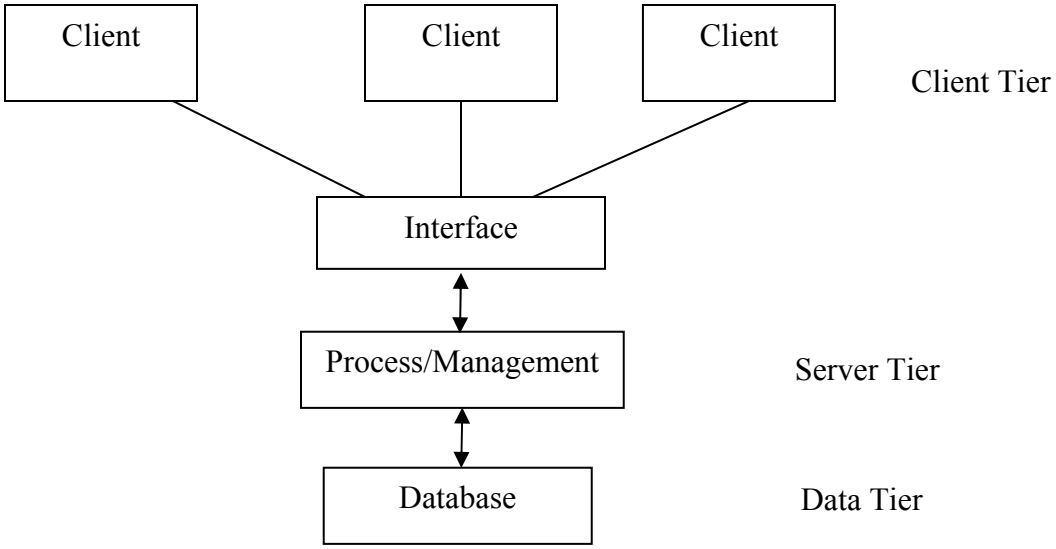


Figure 5.1 The Architecture of PDM-CSP

5.3 General Implementation Methodology

Currently, the interface layer was implemented using Java Server Page (JSP) and Servlets. JSP and Java Servlets were used to create the html web pages. In the server layer, CSP was implemented using Java Constraint Library 1.0 (JCL). JCL was one of the first software libraries to bring CSP to the java world. This library was an open source library developed by JCL team in the Swiss Federal Institute of Technology [JCL, 2000]. JCL provided an Application Program Interface (API) for working with constraint satisfaction problems. These problems included the classic, soft CSP, and continuous CSP. The database layer was implemented using the text file that could be accessed by any programming language. Furthermore, currently the implementation of the PDM system was very pre-matured.

The programming environment is as follows: (1) Windows 2000 professional or other windows systems, and (2) VisualAge for Java from IBM as editor and the web environment.

5.4 Implementation of CSP

The CSP problem representation was implemented based on the following simplifications: (1) each variable corresponds to one component/subsystem, instead of the attribute or feature of a component or subsystem. In this case, if a particular component has two attributes (e.g., the color and size), and the domain of the color is <'Red', 'Blue'>, and the domain of the size is, <'100', '200'>, then the domain of that component or variable corresponding to that component will have 4 models, which are: model 1 ('Red', '100'), model 2 ('Red', '200'), model 3 ('Blue', '100'), and model 4 ('Blue', '200'). So the domain of the variable in this case has 4 models.

The function requirement is represented by the variable too. For example, the capacity of the elevator corresponds to a variable. The domain of the capacity variable was <1000, 1500, 2000, 2500, 3000>. The constraint requirement was also represented as variable.

Care must be taken that since the variable was defined for the whole component, not for an attribute or property of a component (as discussed before), a specification of Preference I may introduce some redundant specification. An example helps to clarify this point. Suppose that the customer has a preference, say the door open style should be "side open" (the customer has no preference on the door speed). The door component has the two features in the current example):

the door speed and the door open style. The codes were developed corresponding to these two features. For example:

ss: single speed;

2s: double speed;

so: side open;

co: center open;

Therefore the code ssso stands for “single speed and side open”.

The constraint requirement as mentioned before is then specified by two unary constraints: Door model {ssso, 2sso}.

The wish requirement in CSP was implemented with Preference II (see the previous discussion in Section 3.6). Currently, only the cost and performance were implemented. This was specifically done by introducing extra two attributes (cost and quality) with each model. As such, the CSP solving process was able to evaluate the total cost and total quality. Appendix B presents a full list of constraints of CSP for the elevator configuration design.

5.5 Implementation of Integration of the Configuration and CAD

The implementation of an interface of the Configurator (i.e., PDM-CSP) was done based on a CAD system: SolidWorks. The interface between the PDM-CSP and the CAD was discussed in Section 4.3.2. In SolidWorks, the elevator system was created. The model was parametric so that the whole elevator product family can be assembled. Fig. 5.2 shows a simplified elevator system.

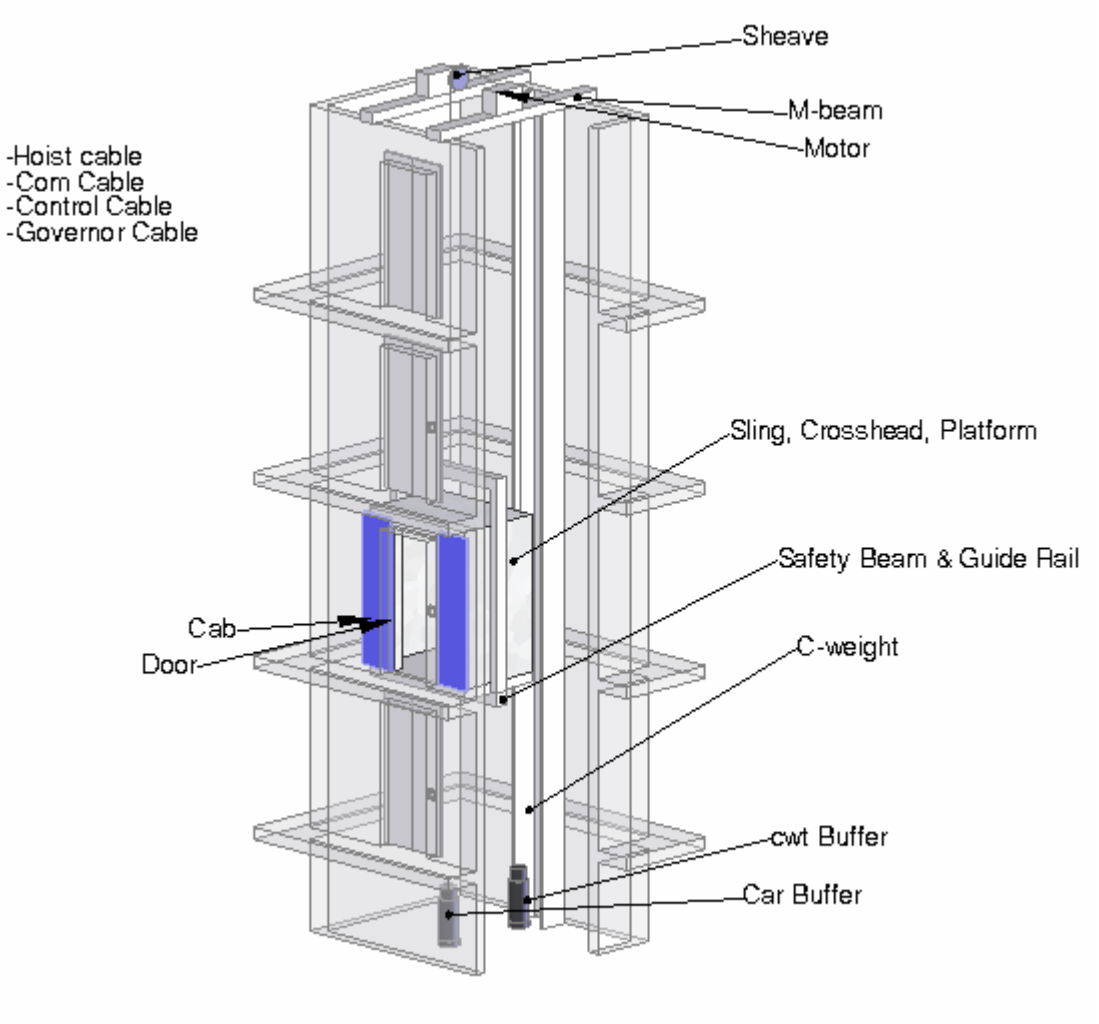
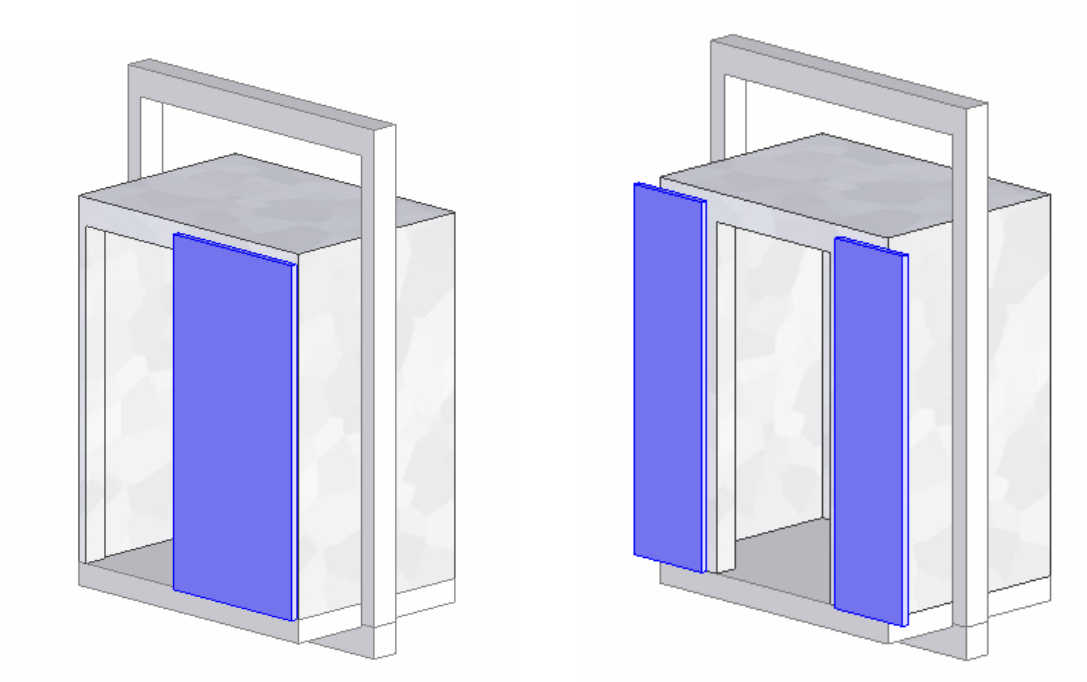


Figure 5.2 Elevator System

Fig. 5.3 shows two configurations of doors. Such an implementation supported the following design scenario. The customer interacts with the Configurator and gets an optional configured product. Then the product is visualized in the SolidWorks environment.



Side-open door

Center-open door

Figure 5.3 Different Configurations for Doors and Car Cabs

5.6 Demonstration

The design problem starts from the design requirements; specifically, it includes:

(1) Function requirements:

(1a) The capacity of the elevator should be 2000 lb,

(1b) The speed of the elevator should be 250 ft/min,

(1c) The car of elevator has a “phone” and a “level indicator”.

(2) Constraint requirements:

(2a) Door opening type should be “Side”,

(2b) Door opening speed should be “Double”,

(2c) Door opening strike side should be “Right”,

(2d) Cab should be “96” inch high, “70” inch wide, and “84” inch deep.

(3) Wish requirements:

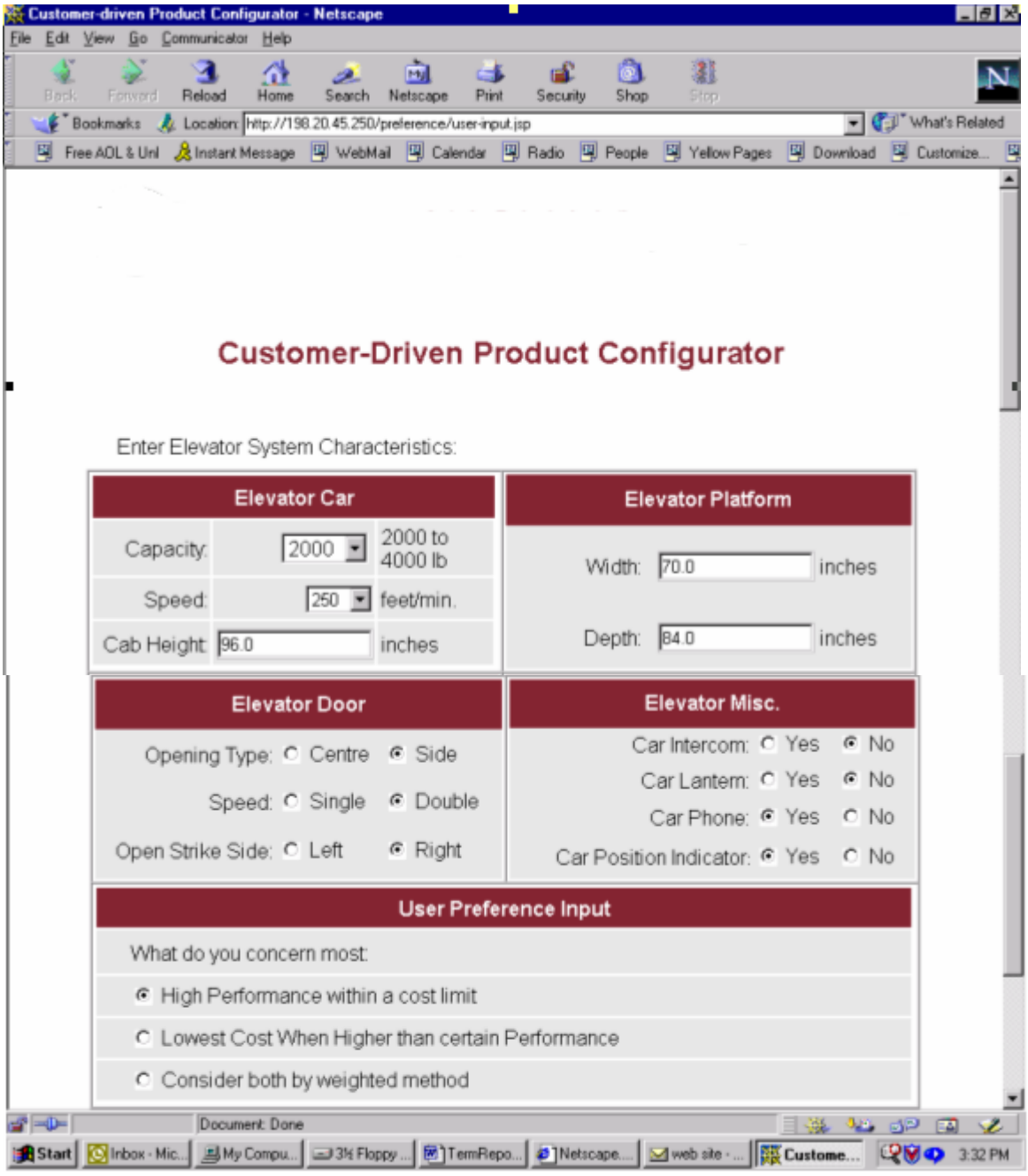


Figure 5.4 Interface for Design Requirement Specification

It is required to have a high performance but within a cost limit. The customer can enter the requirements into the interface, as shown in Fig. 5.4.

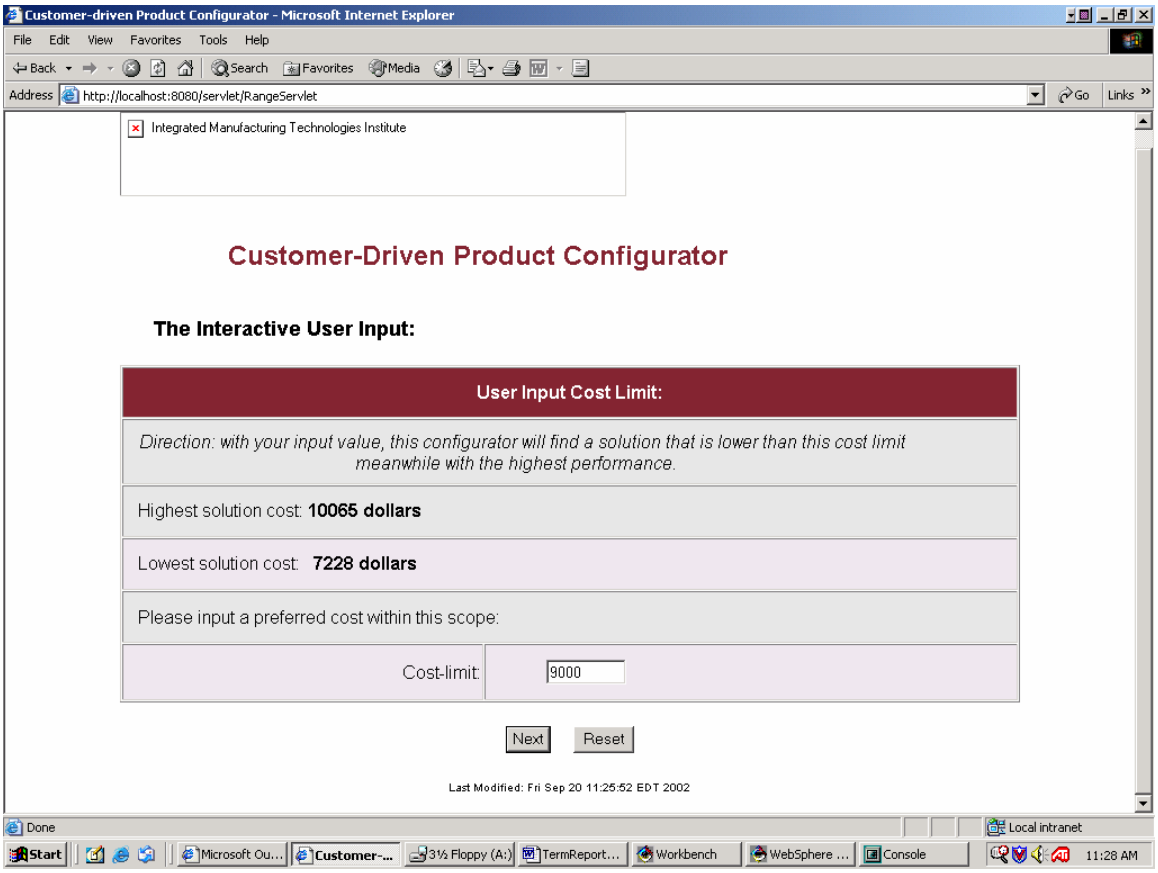
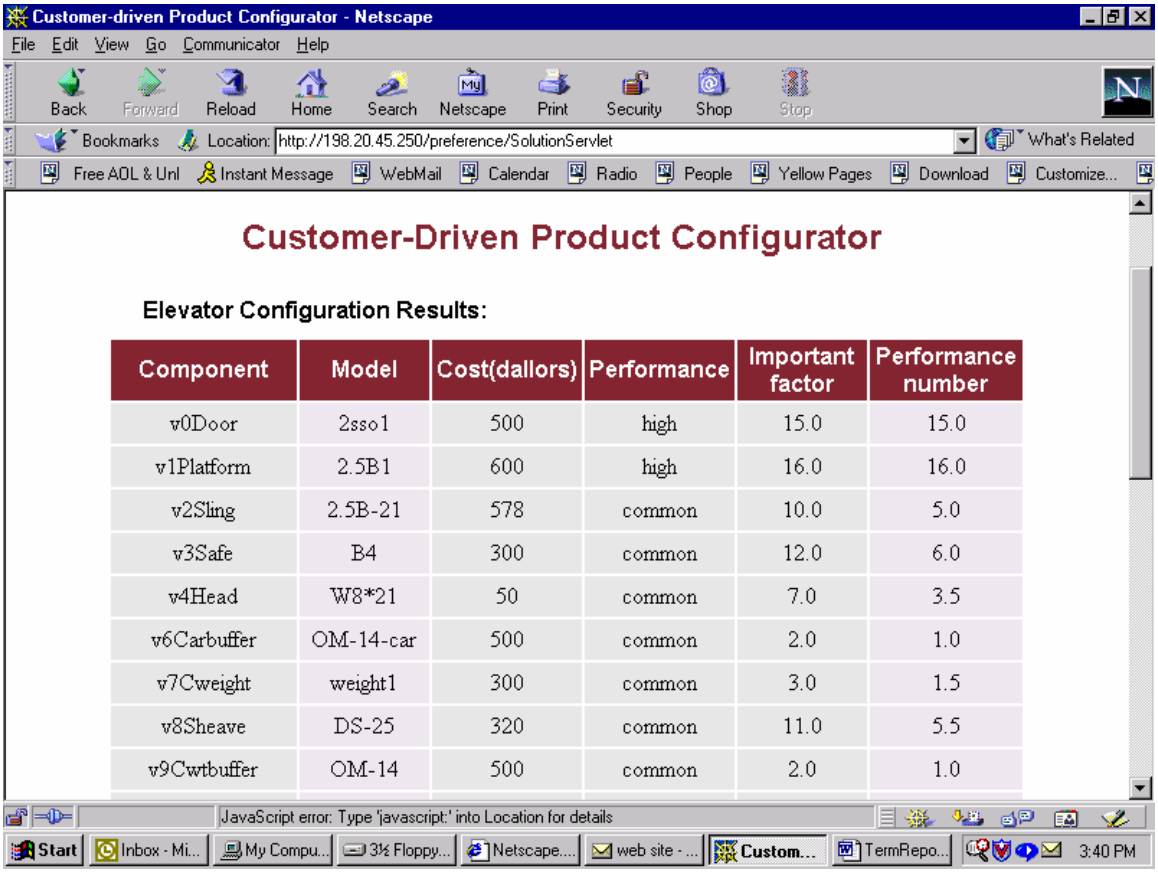


Figure 5.5 Cost Limit Input by Customer

Since the wish requirement is specified as the high performance within a cost limit, the configuration will ask from the customer for the preferred cost information. To facilitate the customer decision making process, the configuration will first calculate the range of the cost limits (Fig. 5.5). After that, the customer can specify a preferred cost which should be within the range, e.g., $9,000 in this case. Finally the Configurator will come up with a solution (i.e., a configured elevator) and shows the solution in the data format of BOM (Fig. 5.6). In this figure,

the configured product's BOM is listed, and each component's model is selected. For example, component door's model is '2sso1', which means a door with "double speed", "side open" features and grade 1 quality. The cost of a model is also attached; for example, the price for model '2sso1' is $500. Some other attributes attached with the BOM are performance (or quality), important factor, and performance number.



Component	Model	Cost(dallors)	Performance	Important factor	Performance number
v0Door	2sso1	500	high	15.0	15.0
v1Platform	2.5B1	600	high	16.0	16.0
v2Sling	2.5B-21	578	common	10.0	5.0
v3Safe	B4	300	common	12.0	6.0
v4Head	W8*21	50	common	7.0	3.5
v6Carbuffer	OM-14-car	500	common	2.0	1.0
v7Cweight	weight1	300	common	3.0	1.5
v8Sheave	DS-25	320	common	11.0	5.5
v9Cwtbuffer	OM-14	500	common	2.0	1.0

Figure 5.6 Solution: BOM for the Configured Product

Chapter 6
Conclusion and Future Work

6.1 Overview of the Thesis

The trend of today's manufacture industry is changing from mass production to mass customization. The companies who win the markets are those who can deliver highly customized products with the fastest speed. One of the strategies to implement the mass customization is to implement the product development into the assemble-to-order (ATO) pattern. As such, the design of a product becomes the determination of a configuration which contains a set of pre-developed components – configuration design for short. In the configuration design, the components will be less engineered. The configuration design problem can be well treated as constraint satisfaction problem (CSP). The matured methods are available for CSP. But the CSP method for configuration design has several limitations. First, CSP requires that design problem be defined at a relatively low level, i.e., the variable, the domain, and the constraint. Second, because of the first reason, the modeling of a configuration design problem into a CSP problem is not convenient. Third, integration of configuration design process based on CSP into life cycle processes (assembly, maintenance, recycling) is difficult because the linkage between CSP model of configuration design and the models for other life cycle processes is not explicitly represented in a data format. So computer processing of the integration is not possible.

This thesis proposed to overcome these limitations by having a product data model wrap a CSP model for configuration design. In this way, configuration design is represented in the product

data model (PDM) level as well as in the CSP level, while the linkage between PDM and CSP is completely looked after by the computer (i.e., automatically maintained). The computer system implemented as such is called the PDM-CSP configurator (or configurator for short). In particular, the following research objectives were defined.

Objective 1: Extend a CSP representation to explicitly incorporating the customer preference and constraint.

Objective 2: Develop an integrated PDM and CSP approach to configuring products.

Objective 3: Develop a framework for integrating a configurator with other product life cycle development systems.

These objectives have generally been achieved. Specifically, an extended CSP model which incorporates two preferences (I, II) was proposed in Chapter 3, which addresses objective 1. The Preference I was in fact a kind of unary constraint. The Preference II was represented into an optimization problem, which was supported by an underlying CSP method. A great deal of discussion in Chapter 3 is on data modeling for the configurator, which includes the data model for configuration design in the PDM level, the data model for configuration design in the CSP level, and the data model for a linkage between PDM and CSP. These data models address objective 2.

Chapter 4 proposed a life cycle configuration design concept and developed a framework for integration of the configurator and other life cycle processes. Specifically, the integration

problem was viewed as the interface problem, i.e., the interface between the configurator and other life cycle processes. For the illustration purpose, two integration examples, an interface between the configurator and CAD systems and an interface between the configurator and SCM systems, were discussed. Integration between two systems involves data integration; data here includes both decision data and its background data (more semantically, data representing decisions on the product development and data representing rationale behind the decisions). It was proposed that the decision data was represented as the structural data, while its background data was represented as the semi-structural data. It was further proposed that the schema-instance data modeling language (e.g., EXPRESS, UML) should be used for the structural data, while the mark up language (e.g., XML) should be used for the semi-structural data. These two data models (structural, semi-structure) share same product ontology; specifically, at the implementation level, the tag in XML should reference to the metadata dictionary which is designed for the structural data model.

Finally, an effort was made to verify the ideas and methods proposed in Chapter 3 and 4. This was demonstrated by an implementation. This implementation includes (1) a CSP system, (2) a configuration design interface on the Internet; (3) an interface between the configuration and the SolidWorks.

6.2 Contributions

A main contribution of the research reported in this thesis is the proposed idea that PDM warps CSP. This idea has laid down a basis for a computer-based and web-based configuration design system. Specifically, there are the following contributions:

(1) Proposed the data models for configuration design, which capture: (i) both the structural and functional information, (ii) the structural and semi-structural data, and (iii) the linkage between the PDM and the CSP. A computer system for supporting configuration design based on this idea allows for a customer-oriented interface and a computationally powerful solver (CSP).

(2) Proposed a framework for integrating the Configurator with other life cycle processes. In this framework, a strategy for integrating the structural and semi-structural data was proposed.

(3) Proposed an implementation of the CSP with a particular reference to the customer preferences.

6.3 Future Work

The study presented here has some limitations or can be further extended. First, in design, it is quite often that information is incomplete (uncertain or imprecise). The configurator needs to be extended to model incomplete information. This can be done by developing the data modeling which enables the modeling of incomplete information [Li et al., 1998]. While at the CSP level, the fuzzy and probabilistic CSP facilities can be applied. It is noted that some conceptual level development about this work has already been done; see [Zhang et al., 2003]. Second, a configuration design problem could involve continuous variables. In this case, the CSP problem becomes a mix of the discrete and continuous variables. The future extension of the PDM-CSP configuration for such a complex CSP problem should be interesting.

References:

Apt, R. K. (1999). "The essence of constraint propagation", *Theoretical Computer Science,* 221(1-2), p. 179-210.

Alting, L. & Legarth, B. (1995). "Life Cycle Engineering and Design", *Annals of the CIRP Vol. 44/2/1995.*

Bartak, R. (2001). "Theory and practice of constraint propagation", *Proceedings of CPDC2001 Workshop* (invited talk), *p.7-14*. Gliwice, June 2001.

Booch, G., Rumbaugh, J., & Jacobson, I. (1999). *The unified modeling language user guide***,** Addison-Wesley.

Bowen, J. & Bahler, D. (1991). "Conditional existence of variables in generalized constraint networks", *Proc. AAAI, pp. 215-220*.

Box, D., Skonnard, A., & Lam, J. (2000). *Essential XML: Beyond MarkUp (The DevelopMentor Series),* Addison-Wesley Pub Co.

Brown, S. (2000). *Customer Relationship Management: A Strategic Imperative in the World of E-Business*. John Wiley & Sons, April 21.

Buxmann, P., Díaz, L., & Wüstner, E. (2002). "XML-Based Supply Chain Management--As SIMPLEX as it is", *Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS'0*2)-v.7, p.168, January 07-10.

Carlson, D. (2001). *Modeling XML applications with UML: Practical e-business application*. Addison-Wesley.

Chen, P. P. (1976). "The entity-relationship model - toward a unified view of data", *ACM Transactions on Database Systems, Vol. 1, No 1,* March 1976, pp 9-36

Codd, E. F. (1970). "A relational model of data for large shared data banks", *Communications of the ACM, vol. 13 No.6* (June, 1970).

Date, C. J. (1990). *An Introduction to Database Systems*, volume 1, 5th edition. Addison-Wesley Publishing Company.

JCL (2000). The JAVA Constraint Library, http://liawww.epfl.ch/JCL/jcl-index.html.

Junker, U. (2000). "Preference-based search for scheduling", *Proceedings seventeenth national conference on artificial intelligence (AAAI-2000).* AAAI press, Menlo Park, CA, USA;.p.904-9.

Junker, U. (2001). "Preference programming for configuration", *International Joint Conference on Artificial Intelligence (IJCAI-2001),* IJCAI-01 Workshop on Configuration, http:// www.ijcai-01.org.

Krause, F., Kimura, F., Kjellberg, T., & Lu, S. C.Y. (1993). "Product Modeling", *Annals of CIRP*, 42:2, 695-706.

Li, Q., Ma, Z.M., & Zhang, W.J. (1998). "Modeling of incomplete information and uncertain information in database for engineering design and production management", *ASME 12th Engineering Information Management Symposium*, Paper number: DETC98/EIM-5686, USA, September.

Lustig, I. & Puget, J. (2001). "Program does not equal program: Constraint programming and its relationship to mathematical programming", *Interfaces Vol 31* No 6: 29-53.

Lyon, D. D. (2000). "On configuration management and program management", http://www.pmforum.org/library/papers/dlyon.htm

Mailharro, D. (1998). "A Classification and constraint-based framework for configuration", *Artificial Intelligence for Engineering Design, Analysis and manufacturing (1998*), *v.12, 383-397.*

Marchal, B. (2000). *Applied XML Solutions-The Authoritative Solution*, SAMS, A Division of Macmillan USA, Indianapolis, Indiana

Marriot, K. & Stuckey, J. P. (1998). *Programming with constraints: An Introduction*, MIT Press, Cambridge, Massachusetts, 1998.

Mesihovic, S. & Malmqvist, J. (2000). "Product Data Management (PDM) system support for the engineering configuration process", *14th European conference on artificial intelligence ECAI 2000 configuration workshop, Aug. 20-25, 2000,* Berlin, Germany.

Miguel, I. & Shen, Q. (2001). "Solution techniques for constraints satisfaction problems: foundations", *Artificial Intelligence Review: 15: 243-267& 269-293*. 2001 Kluwer Academic Publishers.

Mittal, S. & Frayman, F. (1989). "Towards a generic model of configuration tasks", *Proc. Eleventh Int. Joint Conf. Of Artificial Intelligence*, 1395-1401.

Mittal, S. & Falkenhainer, B. (1990). "Dynamic constraint satisfaction problems", *Proc. AAAI Conf., pp*. 25-32.

Rao, S. S. (1996). *Engineering optimisation-theory and practice*, A Wiley-Interscience Publication.

Reeves, C.R. (1991). "Recent algorithmic developments applied to scheduling problems", *Proc. of 9th IASTED Conf. on Applied Informatics*, 155-158.

Sabin, D. & Freuder, E.(1996). "Configuration as composite constraint satisfaction", *AAAI Press.* In workshop notes of AAAI fall symposiums on configuration, 28-36, Menlo Park, CA

Sasikumar, M. (1996). "Case-based reasoning for software reuse", *Knowledge based computer systems-research and applications* (International Conference on Knowledge-Based Computer Systems), pp.31-42, Bombai, India, (December 12-15, 1996). Narosa Publishing House, London.

Shrikhande, S.V. (2000). *On effective information modeling for computer-based configuration management of complex products in manufacturing environments*, M. Sc. Thesis, University of Saskatchewan, Saskatoon, Canada.

Sinha, N. (2004). *Modeling for effective computer support to MEMS product development*, M. Sc. Thesis, University of Saskatchewan, Saskatoon, Canada.

Stumptner, M., Friedrich, G. E., & HaselBock, A. (1998). "Generative constraint based configuration of large technical systems", *AIEDAM (Artificial Intelligence for Engineering, Design, Analysis and Manufacturing)*, special issue on configuration, 12(4), p302-320, September, 1998

Suh, N.P. (1990). *The Principles of Design*, Oxford University Press

Tiihonen, J., Soininen, T., Männistö, T., & Sulonen, R. (1996). "State-of-the Practice in Product Configuration – a Survey of 10 Cases in the Finnish Industry", *Helsinki University of Technology*

Tiihonen, J. & Soininen, T. (1997). "Product configurators – information systems support for configurable products", TAI research center and laboratory of information processing science, Product Data Management Group, Helsinki University of Technology, Finland.

Tsang, E. (1999). "A glimpse of constraint satisfaction", *Artificial Intelligence Review: 13: 215-227*. 1999 Kluwer Academic Publishers, Printed in the Netherlands.

Yost, G. R. (1996). “Configuring elevator systems”, *Int. J. Human-Computer Studies* (1996) 44, p.521-568.

Zhang, W.J. (1994). “An integrated environment for CAD/CAM of mechanical systems”, *Thesis Delft University of Technology*

Zhang W.J. & Werff van der K. (1993). “Guidelines for product data model formulation using database technology”, *Proc. of Int. Conf. on Engineering Design (ICED'93), Vol.3*, The Hague, 1618-1626.

Zhang, W.J., Li, J.X., Xie, H., & Shi, Z.Z. (2003). “A general approach to e-learning software”, *Proceedings of CIE’03: ASME international 23rd Computers and Information in Engineering (CIE) Conference*, September 2-6, 2003, Chicago, Illinois.

Appendix A: XML

Data can be classified as structural data and semi-structural data. Structural data is best described by the so-called "data model" which gives (1) rules to define data structure, (2) rules to define data integrity, and (3) rules to define data behaviour. Database technology is an effective tool for the structural data. The semi-structural data are those that cannot be structured the same manner as the structural data. The semi-structural data may also be called the "document data".

Earlier in 1980s, a language called SGML (Standard Generalized Markup Language) [Shrikhande, 2000] was used to represent semi-structure data in the need of publishing service communities. Later, with the development of the internet technology, the HTML (Hyper Text Markup Language) comes to the stage. From its premise, the HTML is inherently designed for the display of semi-structural data. The main problem with the HTML is its device (display device) dependency. XML (eXtensible Markup Language) [Box et al., 2000] is designed to overcome the device dependency problem with the HTML, which is the premise of the XML. As such, XML has the system architecture, in which the data representation and the data display are separated (see Fig. A.1). In Fig. A.1, XSL stands for e**X**tensible **S**tylesheet **L**anguage; XHTML stands for eXtensible HyperText Markup Language; XML DTD is Document Type Definition.

The premise of XSL consists of three parts: XSLT, XPath, and XSL Formatting Objects. It is to specify the presentation of a class of XML documents by describing how an instance of the class is transformed into an XML document that uses the formatted vocabulary.

The premise of XHML is to replace HTML. The premise of DTD is to define building blocks of a XML document.

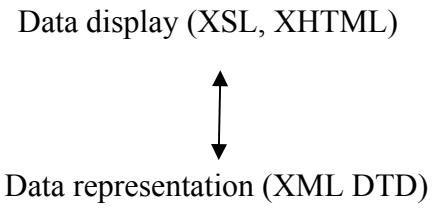


Figure A.1 The XML Architecture

One of the common characteristics of the XML and the HTML is the way to express data, i.e., the concepts of tags and strings. Fig. A.2 shows an example to explain these two concepts. In this example, ‘<note >’, ‘<to>’, ‘<from>’ and ‘<body>’ are called the start tags, and ‘</note >’, ‘</to>’, ‘</from>’ and ‘</body>’ the end tags. Strings are placed between the start tags and the end tags.

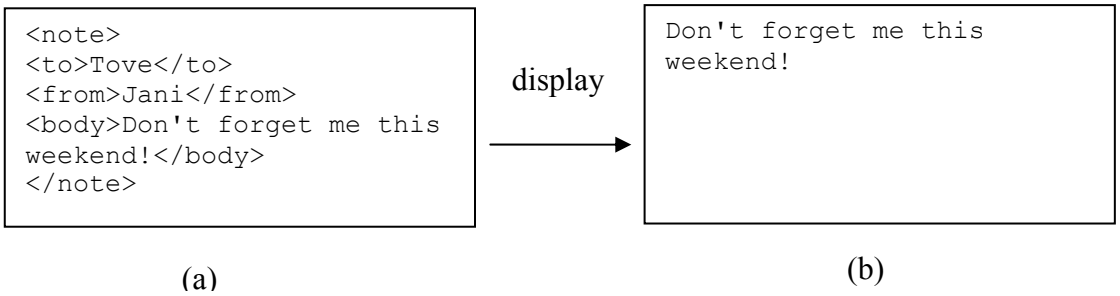


Figure A.2 The Concepts of Tags and Strings

As this thesis focuses on conceptual modeling, the XML DTD (Document Type Definition) is of interest. As mentioned earlier, the purpose of a DTD is to define the legal building blocks of an

XML document. It defines the document structure with a list of legal elements. A DTD can be declared within a XML document with inline format, or as an external reference.

The examples of a XML file using an external DTD are shown in Fig. A.3 and Fig. A.4 respectively. Fig. A.3 is a XML file with an external DTD (the line in bold.), and Fig. A.4 is this external DTD named "note.dtd. It can be seen that the DTD file (see Fig. A.4) defines the tags, such as to, from, heading, and body, specifically to the data type for "to", "from", "heading", and "body".

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM "note.dtd">
<note>
<to>Tove</to>
<from>Jani</from>
<body>Don't forget me this weekend!</body>
</note>
```

Figure A.3 An Example of a XML File With an External DTD Reference

```
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

Figure A.4 An Example of a DTD File

For more details regarding XML, as well as HTML, the readers may refer to

1. XML separates data presentation from data semantics to make XML file machine – independent. Therefore it is a cross-platform, software and hardware independent tool for transmitting information.

2. XML is free and extensible that means you can create your own ‘tags’ (elements for expressing application semantics) and define you own document structures by DTD or XML schema. Therefore it is a powerful tool to be used in variety of industries, communities and research areas.

3. XML is able to manage semi- or un-structured data with some special tools. Unstructured data is the information hidden in a company's e-mails, memos, notes from call centers and support operations, news releases, user groups, chats, reports, letters, surveys, white papers, marketing material, research, presentations, and Web pages. Microsoft and some other vendors (CambridgeDocs: http: // www.cambridgedocs.com/id16.htm) have all developed technology that transforms unstructured data into XML.

In conclusion, XML was created to structure, store and send information, and most importantly, it is a cross-platform, software and hardware independent tool for transmitting information.

Appendix B: Design Knowledge Basefor the Elevator System Design

The variable corresponds to the component of the elevator system. There are 16 components plus 4 optional components. So the number of variables correspondingly to the components is 20. The design requirement is also represented by the variable. There are three design requirements, i.e., “capacity”, “speed”, and “Cab Door Type”. So there are additional three variables corresponding to the design requirements. In total, there are twenty three variables. These variables are listed below:

V_0 | Door

V_1 | Platform

V_2 | Sling

V_3 | Safe

V_4 | Head

V_6 | Carbuffer

V_7 | Cweight

V_8 | Sheave

V_9 | Cwtbuffer

V_{10} | Concable

V_{11} | Hcable

V_{12} | Gcable

V_{13} | Comcable

V_{14} | Machine

V_{15} | Mbeam

V_{16} | Motor

Option variables:

V_{18} | CarPhone

V_{19} | CarLantern

V_{20} | CarIntercom

V_{21} | CarIndicator

And requirement variables:

V_{22} | Capacity

V_{23} | Speed

V_{24} | Door Type

The domains of these variables are listed below:

```
component #Capacity
Domain size #5
value #2000
value #2500
value #3000
value #3500
value #4000
component #Speed
Domain size #5
value #200
value #250
value #300
value #350
value #400
component #DoorOpenSpeed
Domain size #4
value #ssco
value #ssso
value #2sco
value #2sso
component #CarPhone
Domain size #2
value #0
value #1
```

```
component #CarLantern
Domain size #2
value #0
value #1
component #CarIndicator
Domain size #2
value #0
value #1
component #CarIntercom
Domain size #2
value #0
value #1
component #v0Door,15
Domain size #8
value #ssso1,500,high
value #ssso2,400,low
value #ssco1,500,high
value #ssco2,450,low
value #2sso1,500,high
value #2sso2,450,low
value #2sco1,800,high
value #2sco2,780,low
component #v1Platform,16
Domain size #6
value #2.5B1,600,high
value #2.5B2,400,low
value #4B1,700,high
value #4B2,500,low
value #6B1,800,high
value #6B2,650,low
component #v2Sling,10
Domain size #5
value #2.5B-18,554,common
value #2.5B-21,578,common
value #4B-HOSP,680,common
value #4B-GP,699,common
value #6C,899,common
component #v3Safe,12
Domain size #3
value #B1,100,common
value #B4,300,common
value #B6,400,common
component #v4Head,7
Domain size #5
value #W8*8,100,common
value #W8*21,50,common
value #C8*11.5,20,common
value #C10*15.3,40,common
value #C13*16.55,80,common
component #v6Carbuffer,2
Domain size #2
value #OH-1-car,400,common
value #OM-14-car,500,common
component #v7Cweight,3
Domain size #3
value #weight1,300,common
value #weight2,380,common
```

```
value #weight3,450,common
component #v8Sheave,11
Domain size #2
value #DS-20,250,common
value #DS-25,320,common
component #v9Cwtbuffer,2
Domain size #2
value #OH-1,400,common
value #OM-14,500,common
component #v10Concable,3
Domain size #3
value #Light,110,common
value #Middle,120,common
value #Heavy,130,common
component #v11Hcable,4
Domain size #3
value #3-0.5,144,common
value #3-0.625,120,common
value #6-0.625,150,common
component #v12Gcable,6
Domain size #1
value #normal1,300,common
component #v13Comcable,9
Domain size #6
value #Chain1,200,common
value #Chain2,180,common
value #Chain3,165,common
value #Chain4,150,common
value #Chain5,140,common
value #Chain6,130,common
component #v14Machine,17
Domain size #8
value #18-1,1100,high
value #18-2,1000,low
value #28-1,1545,high
value #28-2,1445,low
value #38-1,1855,high
value #38-2,1655,low
value #58-1,1999,high
value #58-2,1799,low
component #v15Mbeam,13
Domain size #10
value #10*25.4,500,common
value #10*35,550,common
value #12*31.8,600,common
value #12*35,650,common
value #12*40.8,700,common
value #12*50,750,common
value #15*42.9,850,common
value #15*50,890,common
value #18*54.7,920,common
value #18*70,950,common
component #v16Motor,18
Domain size #12
value #10HP1,436,high
value #10HP2,336,low
value #15HP1,680,high
```

```
value #15HP2,580,low
value #20HP1,1216,high
value #20HP2,1016,low
value #25HP1,1396,high
value #25HP2,1096,low
value #30HP1,1208,high
value #30HP2,1008,low
value #40HP1,1435,high
value #40HP2,1335,low
component #v17Building,1
Domain size #1
value #normal2,1000,common
component #v18CarPhone,0
Domain size #2
value #0,0,common
value #1,100,common
component #v19CarLantern,0
Domain size #2
value #0,0,common
value #1,120,common
component #v20CarIntercom,0
Domain size #2
value #0,0,common
value #1,250,common
component #v21CarIndicator,0
Domain size #2
value #0,0,common
value #1,120,common
```

Design knowledge is listed below:

Binary constraint:

```
Number of constraints #21
pair0 #v1Platform,v3Safe
Tupledomain size #8
#2.5B1 B4
#2.5B2 B4
#4B1 B4
#4B2 B4
#6B1 B6
#6B2 B6
#4B1 B1
#4B2 B1
pair1 #v1Platform,v2Sling
Tupledomain size #10
#2.5B1 2.5B-18
#2.5B2 2.5B-18
#2.5B1 2.5B-21
#2.5B2 2.5B-21
#4B1 4B-HOSP
#4B2 4B-HOSP
#4B1 4B-GP
#4B2 4B-GP
#6B1 6C
#6B2 6C
```

```
pair2 #v2Sling,v3Safe
Tupledomain size #6
#2.5B-18 B1
#2.5B-21 B1
#2.5B-21 B4
#4B-HOSP B4
#4B-GP B4
#6C B6
pair3 #v2Sling,v4Head
Tupledomain size #5
#2.5B-18 W8*8
#2.5B-21 W8*21
#4B-HOSP C8*11.5
#4B-GP C10*15.3
#6C C13*16.55
pair4 #v3Safe,v6Carbuffer
Tupledomain size #4
#B1 OH-1-car
#B4 OH-1-car
#B4 OM-14-car
#B6 OM-14-car
pair5 #v1Platform,v7Cweight
Tupledomain size #6
#2.5B1 weight1
#2.5B2 weight1
#4B1 weight2
#4B2 weight2
#6B1 weight3
#6B2 weight3
pair6 #v6Carbuffer,v9Cwtbuffer
Tupledomain size #2
#OH-1-car OH-1
#OM-14-car OM-14
pair7 #v8Sheave,v11Hcable
Tupledomain size #3
#DS-20 3-0.5
#DS-25 3-0.625
#DS-25 6-0.625
pair8 #v11Hcable,v16Motor
Tupledomain size #14
#3-0.5 10HP1
#3-0.5 10HP2
#3-0.625 15HP1
#3-0.625 15HP2
#3-0.625 20HP1
#3-0.625 20HP2
#3-0.625 25HP1
#3-0.625 25HP2
#6-0.625 25HP1
#6-0.625 25HP2
#6-0.625 30HP1
#6-0.625 30HP2
#6-0.625 40HP1
#6-0.625 40HP2
pair9 #v14Machine,v15Mbeam
Tupledomain size #20
#18-1 10*25.4
```

#18-2 10*25.4
#28-1 10*35
#28-2 10*35
#28-1 12*31.8
#28-2 12*31.8
#28-1 12*35
#28-2 12*35
#28-1 12*40.8
#28-2 12*40.8
#38-1 12*50
#38-2 12*50
#38-1 15*42.9
#38-2 15*42.9
#38-1 15*50
#38-2 15*50
#58-1 18*54.7
#58-2 18*54.7
#58-1 18*70
#58-2 18*70
pair10 #v14Machine,v16Motor
Tupledomain size #36
#18-1 10HP1
#18-1 10HP2
#18-2 10HP1
#18-2 10HP2
#18-1 15HP1
#18-1 15HP2
#18-2 15HP1
#18-2 15HP2
#28-1 20HP1
#28-1 20HP2
#28-2 20HP1
#28-2 20HP2
#28-1 25HP1
#28-1 25HP2
#28-2 25HP1
#28-2 25HP2
#38-1 20HP1
#38-1 20HP2
#38-2 20HP1
#38-2 20HP2
#38-1 25HP1
#38-1 25HP2
#38-2 25HP1
#38-2 25HP2
#38-1 30HP1
#38-1 30HP2
#38-2 30HP1
#38-2 30HP2
#38-1 40HP1
#38-1 40HP2
#38-2 40HP1
#38-2 40HP2
#58-1 40HP1
#58-1 40HP2
#58-2 40HP1
#58-2 40HP2

```
pair11 #v1Platform,v16Motor
Tupledomain size #24
#2.5B1 10HP1
#2.5B1 10HP2
#2.5B2 10HP1
#2.5B2 10HP2
#2.5B1 15HP1
#2.5B1 15HP2
#2.5B2 15HP1
#2.5B2 15HP2
#4B1 20HP1
#4B1 20HP2
#4B2 20HP1
#4B2 20HP2
#4B1 25HP1
#4B1 25HP2
#4B2 25HP1
#4B2 25HP2
#6B1 30HP1
#6B1 30HP2
#6B2 30HP1
#6B2 30HP2
#6B1 40HP1
#6B1 40HP2
#6B2 40HP1
#6B2 40HP2
pair12 #v7Cweight,v13Comcable
Tupledomain size #6
#weight1 chain1
#weight1 Chain2
#weight2 Chain3
#weight2 Chain4
#weight3 Chain5
#weight3 Chain6
pair13 #v0Door,v1Platform
Tupledomain size #48
#ssso1 2.5B1
#ssso2 2.5B1
#ssso1 2.5B2
#ssso2 2.5B2
#ssso1 4B1
#ssso2 4B1
#ssso1 4B2
#ssso2 4B2
#ssso1 6B1
#ssso2 6B1
#ssso1 6B2
#ssso2 6B2
#2sso1 2.5B1
#2sso2 2.5B1
#2sso1 2.5B2
#2sso2 2.5B2
#2sso1 4B1
#2sso2 4B1
#2sso1 4B2
#2sso2 4B2
#2sso1 6B1
```

```
#2sso2 6B1
#2sso1 6B2
#2sso2 6B2
#ssco1 2.5B1
#ssco2 2.5B1
#ssco1 2.5B2
#ssco2 2.5B2
#ssco1 4B1
#ssco2 4B1
#ssco1 4B2
#ssco2 4B2
#ssco1 6B1
#ssco2 6B1
#ssco1 6B2
#ssco2 6B2
#2sco1 2.5B1
#2sco2 2.5B1
#2sco1 2.5B2
#2sco2 2.5B2
#2sco1 4B1
#2sco2 4B1
#2sco1 4B2
#2sco2 4B2
#2sco1 6B1
#2sco2 6B1
#2sco1 6B2
#2sco2 6B2
pair14 #inputCapacity,v16Motor
Tupledomain size #40
#2000 10HP1
#2000 10HP2
#2000 15HP1
#2000 15HP2
#2000 20HP1
#2000 20HP2
#2000 25HP1
#2000 25HP2
#2500 10HP1
#2500 10HP2
#2500 15HP1
#2500 15HP2
#2500 20HP1
#2500 20HP2
#2500 25HP1
#2500 25HP2
#3000 10HP1
#3000 10HP2
#3000 15HP1
#3000 15HP2
#3000 20HP1
#3000 20HP2
#3000 25HP1
#3000 25HP2
#3000 30HP1
#3000 30HP2
#3000 40HP1
#3000 40HP2
```

```
#3500 25HP1
#3500 25HP2
#3500 30HP1
#3500 30HP2
#3500 40HP1
#3500 40HP2
#4000 25HP1
#4000 25HP2

#4000 30HP1
#4000 30HP2
#4000 40HP1
#4000 40HP2
pair15 #inputSpeed,v16Motor
Tupledomain size #36
#200 10HP1
#200 10HP2
#200 15HP1
#200 15HP2
#200 20HP1
#200 20HP2
#200 25HP1
#200 25HP2
#250 10HP1
#250 10HP2
#250 15HP1
#250 15HP2
#250 20HP1
#250 20HP2
#250 25HP1
#250 25HP2
#300 15HP1
#300 15HP2
#300 20HP1
#300 20HP2
#300 25HP1
#300 25HP2
#350 15HP1
#350 15HP2
#350 20HP1
#350 20HP2
#350 25HP1
#350 25HP2
#350 30HP1
#350 30HP2
#400 25HP1
#400 25HP2
#400 30HP1
#400 30HP2
#400 40HP1
#400 40HP2
pair16 #inputCarOpenSpeed,v0Door
Tupledomain size #8
#ssco ssco1
#ssco ssco2
#2sco 2sco1
#2sco 2sco2
```

```
#ssso ssso1
#ssso ssso2
#2sso 2sso1
#2sso 2sso2
pair17 #inputCarPhone,v18CarPhone
Tupledomain size #2
0 0
1 1
pair18 #inputCarLantern,v19CarLantern
Tupledomain size #2
0 0
1 1
pair19 #inputCarIndicator,v21CarIndicator
Tupledomain size #2
0 0
1 1
pair20 #inputCarIntercom,v20CarIntercom
Tupledomain size #2
0 0
1 1
```