DESIGN INFORMATION HANDLING IN A KNOWLEDGE BASED INTELLIGENT DESIGN SYSTEM

QUANGANG YANG
CARL REIDSEMA

School of Mechanical and Manufacturing Engineering, The University of New South Wales, Sydney, New South Wales, Australia

The advancements in internet technology have had a tremendous impact on the development of knowledge-based engineering systems that support concurrent engineering. Not only the conventional knowledge acquisition and representation techniques need to be improved to adapt the new situation, but the information retrieval and distribution have arisen to be new questions to answer. This article addresses these issues of management and handling of knowledge in a knowledge-based intelligent design system. A search algorithm based on the computation of a similarity index is proposed to retrieve a design case from the project library. An initial design report is used to distribute design information so that the information such as a designer's intent which cannot be included in a standard CAD file, can be retained. In addition, a simple constraint definition frame is presented to define the relationships between critical design parameters, and two information representation schemas, information matrix and constraint tree, are described.

The presented research efforts in this article aim to provide promising tools to harness the full potential of ontologies in knowledge management within an intelligent design system.

Address correspondence to Quangang Yang, School of Mechanical and Manufacturing Engineering, The University of New South Wales, Sydney, New South Wales, 2052, Australia. E-mail: qg_yang@hotmail.com
INTRODUCTION

The increasing competition in today’s global marketplace has motivated companies to use computer-based technology to deliver high quality, low cost products with reduced lead times. New product development strategies are being heavily investigated through the development of Knowledge-Based Engineering (KBE) systems. As a branch of Artificial Intelligence (AI) in the context of product design, KBE can be defined as an enabling technology that allows organizations to capture, structure, and use knowledge about a design and its design process as well as for defining engineering methods and procedures (Calkins 1996).

KBE is a methodology that bridges the gap between knowledge management and design automation (Prasad 2005). By means of incorporating the engineering knowledge that drives the product design process, KBE enables the computer to assist in generating design variants of a product and to automate repetitive design tasks, thus reducing both time and cost in product development. The traditional design automation approaches, however, lack the dynamic nature of product development formulation (Prasad 2005). Whenever any aspect of a product model changes, the computer source code needs to be updated. Therefore, a higher level intelligent system, which is facilitated by KBE, is essential to realize automation in design.

An Intelligent Design System (IDS) proposed by the authors (Yang and Reidsema 2006b) is such a system aiming at design automation, especially in the parametric and detail design phases. IDS is a framework to automate, or aid, decision making associated with design projects. Such a framework includes the methods used in all stages of the problem solving process, including initial problem definition, knowledge acquisition and representation, design problem decomposition, design space characterization, multi-objective optimization, and final solution development. The IDS presents an advanced, strategic approach for realizing the concurrent product development process. On the one hand, an IDS can integrate available computer resources, such as Computer Aided Design/Manufacturing (CAD/CAM) tools (CATIA™, Pro/E™), Finite Element Analysis (FEA) packages (MSC.Patran/Nastran™, etc.), Matlab™, for example, within one system. On the other hand, it provides a platform upon which design team members can communicate with each other to coordinate and integrate their efforts effectively.
This article focuses on the problem of knowledge handling in IDS. After reviewing the current situation in knowledge handling, a problem definition is briefly introduced in terms of the information processing stages in an IDS. Then, information handling activities, including retrieval, distribution, acquisition and representation, are discussed in detail using a simple shaft design example.

CURRENT SITUATION IN KNOWLEDGE HANDLING

The reuse of knowledge is a critical problem in reducing the development costs of knowledge-based systems. A case-based approach is a useful method to represent previous knowledge modules. However, the pure case-reasoning system based on this approach often leads to relatively poor solutions (Günter and Kühn 1999). A promising method is to embed the case based approaches in a knowledge-based design system so that a new design problem can be solved by manipulating new inputs, constraints, and objectives on the basis of previous similar design data. To achieve this integration, all design cases need to be stored and represented as knowledge in a structured manner for reuse, and an effective searching algorithm for previous cases must also be devised.

In the recent study of knowledge systems, knowledge acquisition and representation is widely investigated. Knowledge acquisition and representation is thought to be one of the principal elements of AI, and a critical part of problem solving process (Newell 1982). Knowledge acquisition is the process of extracting knowledge so that it can be used by a problem solving system. Some researchers treat the knowledge acquisition process as a model construction process to facilitate the collection and sharing of knowledge bases and representational tools (Studer, Benjaminis, and Fensel 1998; Clancey 1992). A number of methods have also been developed to automatically or semi-automatically perform the task (Vlaanderen 1990). However, these automatic methods suffer from mismatch difficulties, such as knowledge representation mismatch between human and machine (Vlaanderen 1990), and therefore, they are not applied universally. The process of knowledge acquisition is also technically challenging and time consuming. Knowledge acquisition is primarily carried out using manual methods, such as expert interviews and hence, is seen as a bottleneck in the development of knowledge-based systems (Hayes-Roth 1983).
Knowledge representation involves defining explicit descriptions of the acquired facts and relations in such a way that a computer is able to draw appropriate conclusions by manipulating them (Malhotra 2001). Generally, the representation of knowledge must be adequate, efficient, operational, generalizable, explainable, debugable, and extensible (Woods 1986; Gruber 1989). However, the most important requirement is that the representation is understandable to the user, and capable of being executed by a machine (Gruber 1989; Clancey 1992; Szykman, Sriram, Bochenek, Racz, and Senfaute 2000). There is a large body of research towards knowledge representation in the AI community. Methods for knowledge representation such as rule-based systems, frames and scripts, constraints and facts, and knowledge representation and object-orient languages (Garcia and Chien 1991; Morik, Wrobel, Kietz, and Emde 1993), have been developed. In the context of mechanical design, Gero (1990) introduced a knowledge representation schema referred to as design prototypes. A design prototype stores knowledge including relational knowledge, qualitative knowledge, computational knowledge, constraints, and context knowledge. Gardan and Gardan (2003) proposed capturing knowledge from experts utilizing CAD tools that can then be invoked within CAD software in the form of scripts. Such design scripts can separate the knowledge from the implementation, and effectively bridge the gap between design and knowledge management.

The current studies on knowledge acquisition and representation mostly focus on the conceptual design phase. This is primarily because the identification of high-level function structures in conceptual design requires the knowledge and creativities of a human designer, whereas the subsequent detail design activities normally involve routine tasks such as selection of parameter values. However, bearing in mind that most design problems in industry are incremental, addressing knowledge management in detail design is worthwhile and inevitable in order to realize design automation in detail design. KBE systems are also increasingly used as supporting tools to realize concurrent product development. For Concurrent Engineering (CE) design, the knowledge of interest is mainly concerned with design requirements and constraints (Shakeri 1998), which normally exist within the design coordination problem of multi-disciplinary team design. Therefore, an acquisition and representation schema, which is suitable for this situation, is crucial to implement a CE-enabling knowledge system.
The current situation in knowledge handling has been complicated by the advancements in internet technology. In today’s CE product development environment, members of a multifunctional team may perform a collaborative design task in a distributed environment in terms of space and time. Reidsema and Szczerbicki (2001) presented an agent-based framework, which has a blackboard database interaction structure, to deal with a distributed, collaborative problem. However, the primary aim of their system was to support the CE design process planning. In the development of a distributed design system, Wang (2003) has pointed out that there are currently two major issues. One is that the current information that resides within a product model that is shared and transferred in current design tools is composed of both static geometric information and administrative information. Other information that contains the designer’s intent such as constraints and other dynamic relationships is lost. Another issue is the lack of an information infrastructure that supports an internet-based product development system. Therefore, new questions arise about how to distribute information among design team members in a complete and effective way. Moreover, the information acquisition techniques also need to be improved to adapt to a distributed environment. Traditional techniques such as face-to-face interview and brainstorming have to be replaced by new electronic methods that leverage existing work tools such as Email.

**PROBLEM DEFINITION**

In artificial intelligence, the words data, information and knowledge are often used indistinguishably though their definitions are actually slightly different from each other. In this article, unlike in the most of the computer based design systems, the term information, rather than knowledge, is used. The reason for using information is that in an IDS, the information obtained from the members of the multidisciplinary team is often more direct and concrete, such as requirements and constraints on a specific problem. This information may be different from the traditional knowledge obtained from experts and text books, which is more general and abstract.

From the point view of information handling in IDS, the process can be divided into four stages. The activities involved in the four corresponding stages are:

1. Design case retrieval,
2. Initial model distribution,
Design case retrieval refers to a process of retrieving a similar design case for a new design task from the structured design project library. Two key aspects related to the retrieval are the structural design of the project library and the algorithm used in searching. An initial design model can be established on the basis of the retrieved model and the new design inputs. This initial model, together with its relevant information, is then distributed to the members of the multifunctional design team. Once received, the team members will review the initial design model carefully. Subsequently, the members will present their respective requirements based on their individual information and expertise. Finally, all the information including the requirements gathered from the team members will be represented in an appropriate manner so that it can be used by the IDS during the problem solving process.

In the following sections, these four activities in our proposed IDS will be discussed, using a simple shaft design problem as shown in Figure 1. This shaft consists of four sections: two bearing sections, one driving gear section and one driven gear section. In general, the objectives of a shaft design relate to performance that includes determinations on allowable shear stress, stiffness/mass ratio and cost.

**DESIGN CASE RETRIEVAL**

In an IDS, there are three ways to introduce an initial design model into the system. The first is to search for a previous similar design example from the project library. The second is to browse for an existing model.
The last option is to create a new trial geometrical model for the design problem. Most design problems can be retrieved from previous design problems that have a high degree of similarity, since approximately 75% of design work in industry is thought to consist of either the adaptive or variant type (Singh 1996). As such we assume in this article that a similar design case always exists and the last two methods are not discussed here.

The project library for storing previous design cases should be well organized and indexed to enable an efficient search for a similar design case. A convenient way to do this is to organize the library using the Microsoft Window file system. When a finished design is saved into the project library, it will be allocated to an appropriate family directory from a list of existing directories. Alternatively, a new directory can be created. The name of the family directory must be meaningful and descriptive to describe the nature of the model clearly so that the user can quickly pinpoint the target directory for saving and searching. For example, with the above mentioned shaft, a good family name can be defined as OutstretchingDriveShaft_FourSection_Twogears, meaning that all shafts in this family are out-stretching drive shafts with four sections, and can have two gears mounted.

Another important aspect which affects the efficiency and accuracy of the searching process is how to name the stored file. There are several coding systems developed to identify the design and manufacturing attributes of mechanical components (Groover and Zimmers 1984; Rembold, Nnaji, and Storr 1993), such as the best known Opitz coding system which is used to develop part families (Opitz 1969). Here, a name coding protocol is proposed to name the stored design cases. In this coding protocol, the name has a chain structure, consisting of numbered sets. The numbers represent the values of critical factors which are chosen to represent the component. For a steel shaft as shown in Figure 1, if the length and diameter of Section 2, which are 120 and 30 mm respectively, and the Stiffness/Mass ratio, which is 1696, are chosen as the naming factors, it will have a name 120-30-1696. It should be pointed out that the selected naming factors must reflect the design’s attributes. For example, the objective Shear Stress cannot be a naming factor because its value depends on the external load applied to the shaft. It can be seen that using the values of the critical factors to name a design case is simple and straightforward. However, this coding protocol is a little inflexible though the user is free to select naming factors during the creation of the design case family. One family can only have one protocol; once the
name protocol for a family is determined, any subsequent change to the protocol will cause all the design cases in the family to be renamed.

The search algorithm, which is used to search for a previous similar design example from the project library, depends on the structure of the library. A simple searching algorithm based on the computation of a weighted similarity index is given following our proposed naming protocol. In search, the targeting directory and the name code must be provided, and the weight numbers and the preferred directions of individual factors need also be defined, as shown in Figure 2. The weight of a factor indicates the relative emphasis placed on it during the search process. As for the direction, “=” implies that a closer value to the target is more desirable, while “>=” or “<=” suggests that a greater or lower value is wanted. For a stored design case, its weighted similarity index is computed by comparing the values of the individual naming factors in its name code with those values in the target code. The computation of the relative similarity of an individual factor depends on its preferred direction, thus three different cases can be discussed:

Case 1: If the direction is “=,” the relative similarity is calculated as:

$$s = 1 - \left| \frac{t - e}{t} \right|,$$  \hspace{1cm} (1)
where \( s \) means the similarity, \( t \) is the value indicated in the target code, and \( e \) is the value indicated in the name code of the stored design case. A higher \( s \) suggests that \( t \) and \( e \) are closer. The maximum of \( s \) is 1, meaning that \( t \) and \( e \) are the same.

Case 2: If its direction is “\( \geq \)”, a comparison between \( t \) and \( e \) needs to be performed. If \( e \geq t \), then \( s = 1 \), meaning the target is met. Otherwise, it has:

\[
s = 1 - \frac{e - t}{t}.
\]

Case 3: If its direction is “\( \leq \)”, a similar comparison between \( t \) and \( e \) is necessary. If \( e \leq t \), then \( s = 1 \). Otherwise,

\[
s = 1 - \frac{t - e}{t}.
\]

The weighted similarity index of the stored design case is:

\[
S = \sum_i w_i s_i,
\]

where \( S \) stands for the similarity index, \( i \) is the index of the naming factors, and \( w_i \) are the corresponding weight numbers. For example, for a target name code 125-28-1600, if the weights for the three factors are 2, 1, and 3, and preferred directions are defined as “\( = \)”, “\( = \)” and “\( \geq \)” respectively, the similarity index of a stored design case with a name code 120-30-1696 is:

\[
2 \times \left( 1 - \frac{|125 - 120|}{125} \right) + 1 \times \left( 1 - \frac{|28 - 30|}{28} \right) + 3 \times 1 = 5.85
\]

The pseudo code to perform the search process in a directory is shown in Figure 3. After the similarity indices for all stored design cases are computed, the one with the highest index, which means the highest similarity to the target, can be selected. This method is simple and efficient. Another advantage lies in the use of the weighted sum because the user can put different emphases on different factors by giving different weight numbers. If one factor is not of concern to the designer, then a zero weight number can be given to ignore this factor in the search process.
INITIAL MODEL DISTRIBUTION

Once the initial product model is introduced into the system, a design report can be created to summarize the initial model. Figure 4 shows the first page of the report of the initial product model for the shaft design example. The designer can decide what information he/she wants to include in the report. The presented information can be in various
types including text, data, chart, drawing, table, picture, etc. The right hand side of the window consists of three parts:

1. The controls on the top are used to help the viewer to read the report.
2. The list in the middle is used to select the recipients of the report.
3. The commands at the bottom are used to perform relevant actions.

The initial design report is generated automatically by the system. Currently in our IDS, a predefined form of the report is created in advance. The system can complete the report at run time based on the inputs of the user. This method is very simple, flexible and easy to use.

The initial design report complements the CAD model because it presents information which cannot be included in a CAD file. The report is then sent to the selected design team members along with the CAD file which can be either in the original format such as ‘.CATPart’ for the CAD package CATIA™, or in standard neutral format such as IGES (Initial Graphics Exchange Specification). With such a report, the other designers can quickly ascertain the design rationale contained within the initial product model, and thus make their contributions or requirements to the initial design. In general, the report must include correct and complete information in order for the other designers to have a correct and comprehensive understanding of the initial design model.

**INFORMATION ACQUISITION**

After reviewing the report, the receivers can then propose their respective requirements about the initial design model based on their individual information and expertise. They may modify the variation ranges of design parameters, or even to add or remove critical parameters or design objectives. Generally, they can define constraints which describe the relationships between critical design parameters.

A sample constraint definition frame is configured in IDS, as shown in Figure 5. A member of the design team can define his/her requirements using such a window, and send the requirements back to the IDS. He/she needs to input his/her their name to identify whom a constraint is defined by. He/she also needs to select the type of the constraint being defined. Depending on the selection of ‘if-then’ or ‘Non if-then’ type, the constraint content window changes correspondingly.
A constraint can be defined in respect to either the values or the variations of the parameters. However, for a specific parameter, its value and variation cannot be included in the same constraint. This is because the value and variation of a parameter affect each other; the change of one resulting in a change of another. For a constraint, the indices of all its involved parameters will be recorded. In addition, each constraint has a unique name, and this name will appear in a list box once its definition is confirmed. The name of a constraint includes information such as who defines this constraint and which parameters this constraint involves. The user can also modify or delete an existing constraint by selecting it in the name list.

Once all the constraints have been defined, the IDS will check them to eliminate duplicated items, and to detect conflicting constraints. However, at this moment, this crosschecking process is still very coarse. A difficulty that exists with the crosschecking is that a constraint always has multiple variant expressions. For example, a very simple expression

![Figure 5. Constraint definition frame.](image.png)
\[ a - b \geq 0 \] can have seven equivalent variants, such as \( a \geq b \) and \( b - a \leq 0 \). Thus, how to effectively judge the variants of a constraint remains an area for further research.

**INFORMATION REPRESENTATION**

Information representation involves defining explicit descriptions of the acquired facts and relations by organizing information so as to facilitate decision making in intelligent systems. In this section, the classification of information is first discussed, and then two organizational structures of information are described.

**Classification of Information**

As discussed in our previous research (Yang and Reidsema 2006a), all gathered information is sorted into three groups in terms of its nature: input data information, constraint information and objective information. These three types of information have different roles in the problem solving process, and also have different expressions.

*Input Data Information.* Input data includes geometric dimensions, material properties, physical attributes and characteristics, as well as production and process data. They are normally expressed in numerical form. The input data information will be varied during the problem solving process, and thus can also be called critical design parameters. For the given shaft design problem, all geometrical dimensions, i.e., lengths and diameters of the four sections, are taken as critical design parameters. It must be noticed that those inputs, which are not taken as variable design parameters, will have a constant value, such as the material properties in this example.

*Constraint Information.* Constraint information includes geometrical constraints, standard, and regulatory demands, and manufacturing and process limitations. Constraints must be observed in the design process, and are ultimately reflected in parameter selection. They may generally be classified into two types in terms of their influence on design parameters: the first defines the relationships between the different parameters, and the second type represents limitations on only a single parameter (Yang and Reidsema 2006b). The constraints can be expressed in different
forms, as classified in Figure 6. It should be noted that for an inequality constraint, a greater-than (>) or less-than (<) expression is not acceptable because such an expression is discrete. In addition, a bidirectional constraint means that all parameters involved in this constraint affect each other. Conversely, a unidirectional constraint has a dependent parameter, and the other parameters in the constraint will have an effect on this dependent but they will not affect each other. In an ‘if-then’ constraint, however, parameter(s) in the ‘then’ statement will not affect the parameter(s) in the ‘if’ statement, even for a bidirectional constraint. During the definition of constraints, once the unidirectional type is selected, a list box for selecting the dependent parameter will be automatically enabled as indicated in Figure 5.

In this demonstration of shaft design, six representative constraints are defined, as indicated in Table 1. The first constraint indicates that the two bearing sections (Section 1 and 3) have the same diameters. The second, which involves three parameters, is a space limitation on the distance between the two bearings. The third means that Section 1 must be at least 3 mm longer than Section 3. The fourth is a unidirectional
constraint in which the variation of the diameter of Section 2 is the dependent. The fifth indicates that the total length of Section 3 and 4 remains unchanged, with the last one indicating a conditional constraint defining the relationship between the diameter and the length of Section 4.

Constraint information constitutes a critical component of the knowledge. An expert’s knowledge and expertise are mostly embodied in constraints. In addition, the problem solving process is actually guided by the constraints. In case a design problem fails to be solved, one option to recover the solving process is to employ a constraint relaxation method to loosen those constraints that are negotiable or soft.

**Objective Information.** Objective information includes certain targets and goals that the design is expected to achieve. Design objectives suggest the primary problem solving focus, and are used as indicators to evaluate design performance. Objectives should be clearly stated and uncomplicated. An ambiguous or ill-defined design objective can easily result in either a failure to arrive at a solution or an excellent, but incorrect recommendation.

For the shaft design, three design objectives are identified, namely, maximum shear stress, stiffness/mass ratio, and minimum cost. The first two are actually the most salient objectives in realistic shaft design because they affect safety, material selection, rigidity and vibration characteristics, while the cost is always significantly concerned in product design.

**Organization of Information**

Organization of information involves storing and structuring the information in a manner such that the system can use it efficiently and
correctly. The information in such a structure should be straightforward and be easily understood by both the user and the computer. Moreover, the stored and structured information must be separated from the other parts of the system. Thus, the information can be changed or replaced without altering any other codes, and the system is more compatible to different design problems.

In the development of the IDS, we propose two information representation schemas. One is an information matrix in which all information is organized in a single matrix. Another schema is the constraint tree which focuses on the parameters’ interrelationships.

**Information Matrix.** An information matrix, as shown in Table 2, includes all three types of information. The critical parameters are listed in both rows and columns. The parameters in the columns are treated as the primary parameters while those in the rows are thought of as dependent parameters. Within the matrix, there are four main parts. The first part includes the original values of all design parameters. The lower and upper boundaries of the parameters account for the second part. It can be seen that their variation ranges are given symmetrically with respect to their original values, although the ranges are not necessarily symmetrical. The third part consists of the constraints which indicate how the changes of the column (primary) parameters affect the row (dependent) parameters. The fourth part consists of the dependencies between design parameters and objectives. They are qualitatively indicated by a star, and the exact quantitative relationships will be determined in the later problem solving process.

Embedding the constraints in the information matrix must abide by a few rules. First, all the expressions must be arranged in a way that the involved design parameters must appear on the left of the operators (−, −=, −≤), and the right position can only be occupied by a number. For example, the first constraint in Table 1 is expressed as $\frac{\text{DiaSec1}}{\text{C0}} - \frac{\text{DiaSec3}}{\text{C0}} = 0$ in Table 2. This rule helps to reduce the variants of an expression dramatically. Second, the proportional relationship between the variations of two parameters is indicated by a number which suggests the proportional strength. For instance, the fourth and fifth constraints in Table 1 are implied by two numbers 2 and $\frac{1}{C0}$ respectively in Table 2. We assume that most of the quantitative relationships among design parameters define how the change of a parameter causes the proportional change of another. A rule in IDS is thus formulated that if a
<table>
<thead>
<tr>
<th>Para Name</th>
<th>Ori. Val</th>
<th>Lower B'dary</th>
<th>Upper B'dary</th>
<th>LenSec1</th>
<th>LenSec2</th>
<th>LenSec3</th>
<th>LenSec4</th>
<th>DiaSec1</th>
<th>DiaSec2</th>
<th>DiaSec3</th>
<th>DiaSec4</th>
<th>Shear K/M Stress Ratio Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>LenSec1</td>
<td>30</td>
<td>27</td>
<td>33</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>LenSec1+</td>
<td>LenSec1+</td>
<td>LenSec1+</td>
<td>LenSec1+</td>
<td>LenSec1+</td>
<td>LenSec1+</td>
<td>LenSec1+</td>
<td>LenSec1+</td>
<td>LenSec1+</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>LenSec2+</td>
<td>LenSec2+</td>
<td>LenSec2+</td>
<td>LenSec2+</td>
<td>LenSec2+</td>
<td>LenSec2+</td>
<td>LenSec2+</td>
<td>LenSec2+</td>
<td>LenSec2+</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>LenSec3 &lt;= 180</td>
<td>LenSec3 &lt;= 180</td>
<td>LenSec3 &lt;= 180</td>
<td>LenSec3 &lt;= 180</td>
<td>LenSec3 &lt;= 180</td>
<td>LenSec3 &lt;= 180</td>
<td>LenSec3 &lt;= 180</td>
<td>LenSec3 &lt;= 180</td>
<td>LenSec3 &lt;= 180</td>
</tr>
<tr>
<td>LenSec2</td>
<td>120</td>
<td>115</td>
<td>125</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>LenSec1+</td>
<td>LenSec1+</td>
<td>LenSec1+</td>
<td>LenSec1+</td>
<td>LenSec1+</td>
<td>LenSec1+</td>
<td>LenSec1+</td>
<td>LenSec1+</td>
<td>LenSec1+</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>LenSec2+</td>
<td>LenSec2+</td>
<td>LenSec2+</td>
<td>LenSec2+</td>
<td>LenSec2+</td>
<td>LenSec2+</td>
<td>LenSec2+</td>
<td>LenSec2+</td>
<td>LenSec2+</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>LenSec3 &lt;= 180</td>
<td>LenSec3 &lt;= 180</td>
<td>LenSec3 &lt;= 180</td>
<td>LenSec3 &lt;= 180</td>
<td>LenSec3 &lt;= 180</td>
<td>LenSec3 &lt;= 180</td>
<td>LenSec3 &lt;= 180</td>
<td>LenSec3 &lt;= 180</td>
<td>LenSec3 &lt;= 180</td>
</tr>
<tr>
<td>LenSec3</td>
<td>25</td>
<td>23</td>
<td>27</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>LenSec1+</td>
<td>LenSec1+</td>
<td>LenSec1+</td>
<td>LenSec1+</td>
<td>LenSec1+</td>
<td>LenSec1+</td>
<td>LenSec1+</td>
<td>LenSec1+</td>
<td>LenSec1+</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>LenSec2+</td>
<td>LenSec2+</td>
<td>LenSec2+</td>
<td>LenSec2+</td>
<td>LenSec2+</td>
<td>LenSec2+</td>
<td>LenSec2+</td>
<td>LenSec2+</td>
<td>LenSec2+</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>LenSec3 &lt;= 180</td>
<td>LenSec3 &lt;= 180</td>
<td>LenSec3 &lt;= 180</td>
<td>LenSec3 &lt;= 180</td>
<td>LenSec3 &lt;= 180</td>
<td>LenSec3 &lt;= 180</td>
<td>LenSec3 &lt;= 180</td>
<td>LenSec3 &lt;= 180</td>
<td>LenSec3 &lt;= 180</td>
</tr>
<tr>
<td>LenSec4</td>
<td>50</td>
<td>47</td>
<td>53</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>If (LenSec4 &gt;= 49, DiaSec4 &gt;= 20)</td>
</tr>
<tr>
<td>DiaSec1</td>
<td>25</td>
<td>23</td>
<td>27</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DiaSec1 = 0</td>
</tr>
<tr>
<td>DiaSec2</td>
<td>30</td>
<td>28</td>
<td>32</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DiaSec3 = 0</td>
</tr>
<tr>
<td>DiaSec3</td>
<td>25</td>
<td>23</td>
<td>27</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DiaSec3 = 0</td>
</tr>
<tr>
<td>DiaSec4</td>
<td>20</td>
<td>18</td>
<td>22</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>If (LenSec4 &gt;= 49, DiaSec4 &gt;= 20)</td>
</tr>
</tbody>
</table>
cell value in the matrix is detected to be a number, this number is explained by default as a proportional strength between the variations of the related row and column parameters. Third, the expression of an ‘if-then’ constraint must follow the compact one-line syntax of defining ‘if-then’ rules in Microsoft Excel. That is

\[IF (\text{Condition}, \text{Action\_if\_true}, \text{Action\_if\_false}).\]

In general, these rules can help make the semantic analysis, which will be discussed later, much easier.

There are a few points worth noting regarding this information matrix. First, for a unidirectional constraint, it is only included in the related cell in the upper triangle, while a bidirectional constraint is embedded in the related cells symmetrical with the diagonal. For instance, the fourth constraint in Table 1 is included in upper triangle Cell \((\text{DiaSec1}, \text{DiaSec2})\) in Table 2, but not in lower triangle Cell \((\text{DiaSec2}, \text{DiaSec1})\). Next, if a constraint involves more than two parameters, it can be placed in all relevant cells. For example, the second constraint in Table 1, which involves three parameters, is placed in all six relevant cells in Table 2. This approach in handling a constraint with more than two parameters can also be compared that such a constraint is split into multiple constraints; each of which includes two parameters while the others are considered as constants. Last, a cell can accommodate more than one constraint, and they must be separated by a semicolon, as shown in Cell \((\text{LenSec1}, \text{LenSec3})\) and \((\text{LenSec3}, \text{LenSec1})\) in Table 2.

To understand the meaning of a literal constraint, semantic analysis is performed by the IDS. Semantic analysis is a meaning-finding technique in natural language processing. In the IDS, many predefined standard forms are included in the system to enable this analysis. Each constraint item in the information matrix is compared with these standard forms to determine what constraint it is. During the analysis, the operator \((=, \geq, \leq)\) is used as a break point in an expression. The analysis is then carried out only on the left side expression including the operator, while the right side number, which is in the format of text string, is converted to the numeric type directly. The standard forms are written in respect to the primary parameter and the dependent parameter so that they are suitable for different parameters. An example of such a standard form is:

\[\text{“PriPara} - \text{DepPara} \geq,\]

\text{“}
where \textit{PriPara} and \textit{DepPara} stand for primary and dependent parameters, corresponding to the column and row parameters as mentioned above.

This matrix based information representation schema is straightforward, transparent, integrated, and easy to maintain. However, its disadvantages are similarly obvious. First, the constraints may appear in many different expressions, thus setting up the standard forms in semantic analysis becomes a difficult task, especially for a complex design problem. Second, over a large number of the standard forms, the efficiency of the semantic analysis is compromised because the string comparison itself is less efficient. Finally, this schema is less flexible. For a new problem, new constraint expressions may arise, and thus the new standard forms need to be defined and added to the database. To overcome the disadvantages and to handle complex problems, a constraint tree based information representation schema is proposed.

\textbf{Constraint Tree.} This schema focuses on the hierarchical interrelationships between design parameters which are defined by the constraints. In addition, the constraints in literal form are attached to the system at run time by adding a script engine, Microsoft Script Control, to the application so that the semantic analysis is omitted.

In this schema, there are two requirements in the definition of constraints. The first is that the constraints must be written using Visual Basic (VB) syntax, because the IDS is coded using VB, and the literal constraints will be introduced and executed at runtime. The second requirement is for the ‘if-then’ constraints. An ‘if-then’ constraint can only have one ‘if’ statement and one ‘then’ statement. The ‘if’ statement may be an “and” or “or” statement but the ‘then’ statement must always be single. The accepted form is:

\[ \text{If } c \text{ Then } a, \]

where \( c \) stand for conditions, and \( a \) means the action. Hence, a complex ‘if-then’ constraint which is not in this form, it must be equivalently split into simple ones, as indicated in Table 3.

All the defined constraints, including their relevant information, are stored in a Microsoft Excel file in this representation schema, as shown in Table 4. The relevant information includes who defines the constraint, what name the constraint has, what type it belongs to, and which
parameters it involves. The constraint expression in this table is different from that in Table 1 because the parameter names are replaced by the variables in actual coding. Notice that a value of 99 in DirType marks a bidirectional constraint, while all the other values indicate a unidirectional constraint and the index of the dependent parameter is assigned to DirType.

A constraint tree for each design parameter is then established, based on the constraints in which the parameter is involved and its dependents. Figure 7 shows such a constraint tree of parameter LenSec2. The nodes in the tree represent the parameters, while the links connecting parameters represent the constraints. The constraint tree not only provides the visual interrelationships between design parameters, but also forms the base of handling constraints in the problem solving process.

Finally, a constraint module, which consists of constraint functions, is generated. Each constraint has a corresponding function with the constraint name being the function name. Figure 8 shows the constraint module in which the functions of three representative constraints, 2, 4, and 6, are illustrated. During the iterative process of problem solving, once a parameter is selected as a candidate for variation, following its constraint tree, all its constraints and dependent parameters will be evaluated by calling up the relevant function in the constraint module. Inside a constraint function, the first step is to check whether a constraint is activated, if not, a value “Not Activated” is then assigned to the function and the function implementation is terminated. The second step is to check whether a dependent is frozen. A frozen dependent is a parameter that has already appeared in the upper levels of the tree and thus cannot be varied in a lower level. If a dependent is frozen, it cannot be varied, meaning the trial to solve this constraint fails. The last step is to compute the value or variation of the dependent so that the constraint can be satisfied.

<table>
<thead>
<tr>
<th>Original format</th>
<th>Equivalent simple format</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 If c Then a\textsubscript{1} and a\textsubscript{2}</td>
<td>If c Then a\textsubscript{1}, If c Then a\textsubscript{2}</td>
</tr>
<tr>
<td>2 If c Then a\textsubscript{3} Else Then a\textsubscript{2}</td>
<td>If c Then a\textsubscript{1}, If Not c Then a\textsubscript{2}</td>
</tr>
<tr>
<td>3 If c\textsubscript{1}, Then a\textsubscript{1} ElseIf c\textsubscript{2} Then a\textsubscript{2}</td>
<td>If c\textsubscript{1} Then a\textsubscript{1}, If c\textsubscript{2} And (Not c\textsubscript{1}) Then a\textsubscript{2}</td>
</tr>
<tr>
<td>4 If c\textsubscript{1} If c\textsubscript{2} Then a</td>
<td>If c\textsubscript{1} And c\textsubscript{2} Then a</td>
</tr>
</tbody>
</table>
Table 4. Constraint database in IDS

<table>
<thead>
<tr>
<th></th>
<th>Constraint 1</th>
<th>Constraint 2</th>
<th>Constraint 3</th>
<th>Constraint 4</th>
<th>Constraint 5</th>
<th>Constraint 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>From</td>
<td>Manufacturing</td>
<td>Manufacturing</td>
<td>Marketing</td>
<td>Member 1</td>
<td>Member 2</td>
<td>Member 3</td>
</tr>
<tr>
<td>Name</td>
<td>R0_0_O4_O6</td>
<td>R1_0_O0_O1_O2</td>
<td>R2_1_O0_O2</td>
<td>R3_1_V5_V4</td>
<td>R4_1_V3_V2</td>
<td>R5_1_O3_O7</td>
</tr>
<tr>
<td>IfThen</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>ExpType</td>
<td>=</td>
<td>&lt;=</td>
<td>&gt;=</td>
<td>=</td>
<td>=</td>
<td>&gt;=</td>
</tr>
<tr>
<td>DirType</td>
<td>99</td>
<td>99</td>
<td>99</td>
<td>5</td>
<td>99</td>
<td>99</td>
</tr>
<tr>
<td>NonIfThen Para</td>
<td>4_6</td>
<td>0_1_2</td>
<td>0_2</td>
<td>5_4</td>
<td>3_2</td>
<td></td>
</tr>
<tr>
<td>NonIfThenExp</td>
<td>ParaVal(4) =</td>
<td>ParaVal(0) +</td>
<td>ParaVal(0) =</td>
<td>ParaVar(5) =</td>
<td>ParaVar(3) =</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ParaVal(6) +</td>
<td>ParaVal(1) +</td>
<td>ParaVal(2) + 3</td>
<td>2*ParaVar(4)</td>
<td></td>
<td>– ParaVar(2)</td>
</tr>
<tr>
<td>IfPara</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>IfExp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Para Val(3) ≥ 49</td>
</tr>
<tr>
<td>ThenPara</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>ThenExp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Para Val(7) ≥ 20</td>
</tr>
</tbody>
</table>

*Para = Parameter, Val-Value, Var = variation, Exp = Expression, Dir = Direction.*
A constraint module generated by the system is given in text format. However, VB, which is used to code the proposed IDS, is a compiled language rather than an interpreted language. This means that the text code generated at runtime cannot be executed directly because the compilation is not available at runtime. Fortunately, VB is so flexible that the interpreted language, such as scripting language, can be attached to it. In the IDS, Microsoft Script Control, which is an ActiveX component that can be used as a form-hosted control, is employed to introduce the code to the system. By simply placing the Script Control on a VB form and writing the following one line code:

\[
\text{ScriptControl1.Modules(ConstraintModule).AddCode TextConstraint,}
\]

the constraint module can then be attached and executed at runtime.

Figure 7. Constraint tree of parameter LenSec2.
The constraint tree based representation schema is more suitable for complex design problems. Although the schema itself is complex, it makes the problem solving process in the IDS easier and more efficient because the semantic analysis of the literal constraint is omitted. Moreover, the constraint tree based schema is very flexible because the formation of the constraint module is independent of the problem solving process.

Figure 8. Constraint module.
CONCLUSION

This article not only addresses traditional information handling aspects including acquisition and representation, but also puts forward new issues such as information retrieval and distribution. A search algorithm based on the computation of a similarity index is proposed to retrieve a design case from the project library. An initial design report is also devised. Such a report can be generated by the system and sent to the design team members together with the geometrical model, and thus information, including a designer’s intent which cannot be included in a standard CAD file, can be retained. Given that constraint information constitutes a critical component of the knowledge and is ultimately reflected in parameter selection, a simple constraint definition frame is then presented to define the relationships between critical design parameters. Finally, two information representation schemas are described in detail. The matrix based schema is simple, explicit and integrated. The tree based schema is flexible, efficient, and more suitable for complex design problem. These methods provide promising tools to harness the full potential of ontologies in knowledge management within an intelligent design system.

REFERENCES


