NESTED RIPPLE DOWN RULES (NRDR) AS A DESIGN ASSISTANT IN MECHANICAL DIMENSIONAL TOLERANCING

Ramsey F. Hamade  
American University of Beirut  
Mechanical Engineering Department  
P.O. Box 11-0236, Riad El-Solh  
Beirut 1107 2020, Lebanon  
Email: rh13@aub.edu.lb

Ali H. Ammouri  
American University of Beirut  
Mechanical Engineering Department  
P.O. Box 11-0236, Riad El-Solh  
Beirut 1107 2020, Lebanon  
Email: ahe11@aub.edu.lb

G. Beydoun  
School of Information Systems and Technology (SISAT), Faculty of Informatics, University of Wollongong, Wollongong NSW 2522 Australia  
Email: beydoun@uow.edu.au

Abstract  
The dimensional tolerancing knowledge management system presented in this paper uses Nested Ripple down Rules (NRDR) targeted towards incrementally capturing expert design knowledge. A demonstrated example of such captured knowledge is that which human designers utilize in order to specify dimensional tolerances on shafts and mating holes in order to meet desired classes of fit as set by relevant engineering standards. In doing so, NRDR interface was designed to receive mathematical functions with their specifications prior and during the KA process. This is necessary to be able to exploit relationships among several classes with respect to certain numerical features of the cases in order to accelerate the convergence of the NRDR knowledge acquisition process by generating artificial cases which are likely to trigger the addition of exception rules. The incorporation of equations constitutes a novel contribution to the field of knowledge acquisition with NRDR. The developed dimensional tolerancing knowledge management system would help mechanical designers become more effective in the time-consuming tolerancing process of their designs in the future.

Keywords: Knowledge-based systems, nested ripple down rules, NRDR, knowledge acquisition, design, fits and tolerances.

1. INTRODUCTION

Proper dimensional tolerancing is critical to the success or failure of the functioning of mechanical designs. Mechanical systems are represented by parts using geometric primitives, all of which describe ideal shapes. However, actual manufactured parts are necessarily imperfect approximations to those ideal shapes. Therefore, it is necessary to specify tolerancing information during design so that it can be decided whether a manufactured part is acceptably close to the designed ideal during inspection.

While many of the actions during the detailed design process are automated, dimensional tolerancing, involve intense decision-making and, therefore, remains a time-consuming and human-intensive activity in the design and manufacturing processes. Typically, upon building the preliminary design, a designer sequentially spends a significant amount of time re-dimensioning features and ‘annotating’ certain tolerancing information. As correctly pointed out in [1], ‘manual charting is tedious and error prone, hence, attempts have been made for automation’. Radack and Sterling [2] lamented that “the designer is left with the responsibility of ensuring that the tolerances are complete and consistent. The systems do not ensure that tolerances are reasonable or meaningful”. This is certainly true in the case of traditional 2-D drawing-based manufacturing as well as the up-and-coming route of releasing the mechanical database in electronic form [3]. For such a purpose, standards such as ISO 10303 Product Data Representation and Exchange (STEP) Part 47 “Shape variation tolerances” [4] emphasize the proper definition and representation of tolerances.

Knowledge-based systems (KBS) broadly refer to intelligent software programs that apply expert knowledge to the solution of problems. Utilizing computer-based techniques for the purpose of automating tolerance generation and modeling to enhance the process of specifying proper tolerances is an area of active research [5-7]. The idea being is to develop rule-based expert systems to help the designer create complete and functional designs with appropriate dimensions & tolerances in the design stage. For proper dimensional tolerancing, it is this domain of tolerancing expertise that must be captured and represented as a design knowledge base (DKB). Ripple Down Rules (RDR) are used for such a foundational representation in this work.
While most design rationale frameworks such as IBIS, COQ or DRL are focused on the initial development of the knowledge base of a system (see for example the review paper [8]), RDR shifts the development emphasis to maintenance by blurring the distinction between initial development and maintenance [9]. An essential requirement for the development of the KBS is the ease of acquisition and maintenance of the knowledge. RDR [10] is a knowledge acquisition method which proved very successful for developing large knowledge bases for classification tasks [11]. With RDR, knowledge maintenance is a simple process which can be done by the user without guidance of a knowledge engineer [12]. Furthermore, RDR is optimized for maintenance of propositional rule-bases and ensures very high behavior coverage as the systems evolves [13] so that there is no need to import the rule-base in a truth maintenance system (TMS).

In this paper, an approach for acquiring knowledge using RDR for dimensional tolerance is presented. A knowledge base targeted towards capturing expert tolerancing knowledge is built. This system (dubbed DesignAssistant) is demonstrated by specifying dimensional tolerances on shafts and mating holes in order to meet desired classes of fit as set by relevant engineering standards.

2. ACQUISITION OF KNOWLEDGE

In order to construct the knowledge base the design knowledge must be acquired. In this work a spiral process of knowledge acquisition is envisaged, similar to [14] of coming stepwise closer and closer to an operationalization of the knowledge in question. The present approach follows the work on knowledge acquisition which allows knowledge acquisition and maintenance without a knowledge engineer [9,15]. Experts are usually able to explain their reasoning process on a particular problem instance in rather general terms that cover at least the given concrete next step in their design process. However, their explanation may be quite inaccurate in the sense that for other design problems their explanation would not deliver the design step they would actually take. Either their explanation would not cover the step they would take or their explanation would suggest design steps they would not actually consider. Thus, an approach similar to [16] it is pursued, which allows to incrementally acquiring complex concept definitions without demanding an operational definition from the expert. Rather, the expert is merely required to judge whether the concept applies to particular instances. This is a much more natural task for an expert than to articulate general rules on how to judge on any particular instance.

Such an expert design process involves more complex reasoning than just the association of design sequences which were useful in other design instances. For example it involves some causal reasoning on a rather abstract level. However, it seems difficult to devise a general inference mechanism, which could accommodate such expert reasoning. This is particularly the case, since much of such reasoning will not be at a conscious level to the expert. More applicable to tolerancing, the example illustrates that such a dimensional tolerancing knowledge management system can be utilized either to (1) given a class fit, specify tolerances for nominal dimensions or (2) given actual dimensions, back out the corresponding fit. In Section 4 below, these two schemes are referred to as the ‘forward scheme’ and the ‘backward scheme’, respectively.

Ripple Down Rules (RDR) are used as foundational representation for the present workbench. An essential requirement of the workbench is the ease of acquisition and maintenance of the search knowledge. For this purpose, RDR is used as a starting point for the implementation of the KBS and the learning module. An RDR tree is a collection of simple rules organized in a tree structure. Every rule can have two branches to two other rules: a false and a true branch. An example is shown in Fig. 1. When a rule applies a true branch is taken, otherwise a false branch is taken. The root node of an RDR tree contains the default rule whose condition is always satisfied. The root node is of the form “If true then default conclusion”. The default rule has only a true-branch. In RDR, if a ‘true-branch’ leads to a terminal node t and the condition of t is not fulfilled the conclusion of the rule in the parent node of t is taken. If a ‘false-branch’ leads to a terminal node t and the condition of t is not fulfilled, then the conclusion of the last rule satisfied ‘rippling down’ to t is returned by the knowledge base. The knowledge base is guaranteed to return a conclusion as at least the default rule is satisfied ‘rippling down’ to t. Hence the inference is handled implicitly within the structure of the knowledge. When the expert disagrees with the conclusion returned by the knowledge base, the knowledge base is said to fail and requires modification.

Figure 1: An example of ripple down rules.

An important strength of RDRs is the fact that they can be easily modified in order to become consistent with a new case without becoming inconsistent with previously classified cases. This is because every time a rule r is added to a parent rule p, r classifies the case which triggered its addition (the so-called corner stone case) correctly, and excludes all cases which are correctly classified by p. In their simple form, RDRs use simple attribute-value combinations as conditions for the rules [11,12]. When the expert enters a new rule r, s/he chooses the conditions of r from the so-called ‘difference list’ [17]. This list contains attributes satisfied by the case which triggered addition or r, and it excludes all attributes satisfied by any of the cases covered by the parent of r.

3. SYSTEM ARCHITECTURE OF THE DESIGN ASSISTANT SOFTWARE

The architecture of the KA DesignAssistant system is described in Figure 2. A brief discussion follows which explains the function of each subsystem.

1. User interface: This module reads the expert input and displays the system’s answer to a search request. Further, it provides graphical representation of the
knowledge base and graphical output of the automatic assistant to the expert.

2. Knowledge acquisition module: gets the expert input through the user interface. It maintains the knowledge base as well as the knowledge (case) data base.

3. (Search control) knowledge data base: It stores what the expert expresses as search control knowledge. Using this knowledge, given the possible next states from the previous module, this module passes through only those states seen as worth pursuing deeper during the search. This module contains the larger part of the domain knowledge. This knowledge base is built during the actual knowledge acquisition process.

4. Knowledge acquisition assistant: provides hints to the expert to which parts of the knowledge base may need to be modified while ensuring the consistency of the knowledge base with the case database. It relies on past interactions with the expert stored in the case data base to give these hints.

5. Knowledge (case) data base: contains all cases classified by the expert. It allows retrieval of these cases according to their classifications time stamped. Thus, this data base contains a complete history of the interactions with the expert. Although, not all of the interactions affect the knowledge base development, they are essential for the functionality of the knowledge acquisition assistant.

6. Search engine: It controls the generation of the search tree through interactions with the knowledge base. It saves local decisions about search tree nodes in the working memory. It also examines the pruned search tree and chooses an answer according to one of several evaluation criteria set by the user.

7. Domain specific search operators module: contains a set of search operators forming an instance generator.

Given a particular search state, this module can generate all immediate next possible states. This module also allows the knowledge base to interpret any domain specific primitives used by the expert while describing his/her knowledge to the system. Also is where mathematical functions are declared.

8. Working Memory: stores the progress of the search, which is often used by the expert to explain his/her decisions. In electronic circuit design, for example, and solving a component placement problem, a circuit designer chooses his/her next step based on a rough plan; this plan prevails in the progress towards finding a problem solution. Consequently, this progress is also used by the knowledge base to make decisions. The working memory also stores higher order features of steps (i.e. search states) of the evolving search plan. This reduces computational requirements as these features can get used again at a later stage of the search.

9. Dimensional Tolerancing Module: This constitutes the ‘back end’ of the DesignAssistant system and may be a standalone module or a module that interfaces with an existing CAD tool. It contains the part’s engineering description: shape, size (nominal dimensions), dimensional tolerances, and factors of fit, form and function. It inputs nominal dimensions into the intelligent modules of the KA system and retrieves the upper and lower values on the nominal value, i.e., the desired tolerances. Equally feasible is the inverse problem where the input is a pair of existing dimensions one for the shaft and the other for the hole with this module retrieving an applicable class of fit (if present).

4. THE CLASSICAL FIT PROBLEM

A classical example common to the discipline of mechanical engineering by which the relative fit of a shaft to a hole is studied [18]. Depending on the desired mating functionality between a shaft and a hole, Figures 3 and 4 display 31 different classes of interference/clearance fits. These classes cover a wide range of cases varying from loose clearance fit (sliding or running, RC) to tight interference fit (force, FN). In between, there are several variations of locational classes of fit namely: locational clearance (LC), locational transitional (LT), and locational interference (LN). Shown in the figures are limit values – designated L for the lower tolerance limit and U for the upper tolerance limit for the shaft and hole, respectively. In order to calculate the upper and lower tolerance bounds on the diametrical dimension, the limit values L and U are multiplied by the nominal dimension (D, diameter of shaft and hole) raised to a power of 0.333 as follows:

Lower Bound Tolerance = \( L \times D^{0.333} \)

\[ \text{Upper Bound Tolerance} = U \times D^{0.333} \] (1)

The resulting tolerance values have units of mils (1/1000 of an inch). The tolerated dimension is, therefore, arrived at as a bounded value between the lower and the upper tolerance bounds:

\[ \text{Toleranced Dimension} = \text{Nominal Dimension} \times \frac{\text{Upper bound tolerance}}{\text{Lower bound tolerance}} \] (2)

Given a desired class of fit (e.g. LC1, LC2), the scheme gives the upper and lower tolerance bounds for the
nominal diameter of the shaft/hole. This is conventionally described as the forward scheme. The backward scheme, on the other hand, is described as: given actual diametrical dimensions for the shaft and hole, it is desired to correctly identify the resulting class of fit. This latter scheme is demonstrated in the section below where the application in tolerancing is described.

5. NRDR APPLICATION TO DIMENSIONAL TOLERANCING

The classical example of the shaft-in-a-hole mechanical fit problem as introduced above is considered. In the demonstrated 'backward scheme', the user feeds in one desired diametrical dimension for the shaft and another dimension for the mating hole. To this, the software returns a match to one of 31 possible fit criteria. Two functions are needed for the proper evaluation of this hole-shaft fit problem. One function is needed for checking if the actual value of the hole or shaft lies within the limits of a certain class of fit. This first function is based on (1) such that

\[ L \times D^{0.333} < (R - D) \times 1000 < U \times D^{0.333} \]  

where L, U, and D are as defined above and R stands for the actual value of the case for the hole or the shaft. The formula returns true if the inequality is true, otherwise false. The application of equation (3) necessitated that RDR is extended to allow for the utilization of mathematical functions and the corresponding 'quantitative' values. This is a major extension over other applications where the focus of NRDR development has been on 'qualitative' rules (e.g. chess playing [19]). Formally, a domain where each case is described by a set of attributes \( \text{Atts} = \{A_1, \ldots, A_n\} \). Each attribute \( A_i \) assumes a value from its respective domain \( D_i = \{v_1, \ldots, v_k\} \) or \( D_{N_i} = \mathbb{Z} \) (set of integer numbers) in case of a numerical attribute. A rule \( r \) is a conjunction of conditions upon some or all of those attribute values in \( \text{Atts} \) together with a class label that is assigned to a case, if all conditions are satisfied. i.e., a rule is of the form

\[ r = A_{1,i} = v_{1,i} \land \ldots \land A_{n,i} = v_{n,i} \text{ then } c \]  

where \( \land \) stands for any of the relations in \( =, \leq, \geq, >, \# \) and \( A_{n,i} \) has a numerical domain.

The second function is needed to check whether or not the case may be classified as Locational and is an equality check

\[ A = B \]  

This function returns true if string A is equal to string B where A or B may have any value of one of the attributes. Defining a function includes both naming the function and its attributes, as well as specifying the attribute type.

5.1. The Fits Case Generator

The Fits case generator module generates cases of hole, shaft, and diameter to be fed to the NRDR program. These cases are such that each one corresponds to a class of fit. With the exception of case LN1 (which was found to fall completely within the limits of the class designated LT5), the tree was built to contain all cases. The tree has a separate rule for each class.

Fits Case Generator reads from two Text (MS-DOS) files which are created with the desired values in the described format in order to generate the cases. The first input file <Limits.txt> is a file created by Microsoft Excel. It contains the limits of the classes for which cases are to be generated. It also contains the nominal diameter of the case. Figure 5 is an example of cases generated where the nominal diameter = 1 inch (1st column). The 2nd and 3rd columns are the lower and upper limits on the hole, respectively, while the 4th and 5th columns contain those of the shaft. For each class, a case is generated with random hole and shaft values that fit within the given limits. <Locational.txt> is the second file which is also created by Microsoft Excel. It contains only one column. The first row is the name of the cases attribute "Locational", the second row is empty, the rest of the rows, in the order of the limits in the <Limits.txt> file, are the values of the "Locational" attribute, ‘Yes’ or ‘No’.

Fits Case Generator outputs the file < fits_cases.txt> the format of which is similar to that of the cases prepared by Microsoft Excel to be loaded by NRDR. The file contains...
a random case for each class of fit introduced by the file of the limits. Figure 6 is an example of such a file where the first row contains the case attribute names and the rest of the rows contain the cases.

![Example File](image)

Table: Example of the <fits_cases.txt> file (showing only the first 17 fit cases for illustration purposes).

5.2 Fits Domain Construction

Having generated fit cases, a domain (the Fits Domain) was constructed. Defining the domain includes case preparation, and function compilation including specifying the names and types of the relevant attributes. For example, the attribute name Nominal Diameter which represents the nominal diameter of the hole-shaft system is of type 'NUMBER'. Other attributes are Hole (actual hole diameter) and Shaft (actual shaft diameter) which are both of the 'NUMBER' type. Cases generated from the Fits generator module are then loaded. Defining functions such as (3) involves naming the function and its attributes, as well as specifying the type of each attribute. Function attributes include: Lower Limit of a class of fit for the Hole or the Shaft, Upper Limit of a class of fit for the Hole or the Shaft, Real Value of the hole or the shaft, all of which are of the "NUMBER" type. For function (5), attributes A and B are defined to be of the 'String' type. Utilizing Workspace in Microsoft Visual C++, declaring these functions involves an algorithm which results in passing all the attributes to the function. While declaring the function in (3) and in order to declare the condition for the function, two comparisons (joined with the logical AND operator) are made. If the condition is not satisfied, false is returned. Next in the algorithm, function in (5) is similarly declared, thus, completing function declaration.

5.3 The Case Validation Module

Next, the Case validation module performs case validation sequentially starting from the first case of fit designated RC1. The expert is asked whether he accepts the conclusion of the tree. A 'DEFAULT' conclusion corresponds to the first rule of the tree that is always true. This default conclusion only appears if NRDR concludes that no other plausible conclusion exists. Selecting 'Yes' will keep the conclusion resulting in no changes to the tree. Selecting 'No' requires justification of the refusal of the conclusion.

5.4 Adding Rules

A function would have to be selected in order to add a new rule. Figure 7 shows the functions as they appear in the GUI. The first function is needed to add a condition for the size of the hole. The value of an attribute of a function can be a value entered as text or number. The value will be stored in the rule, and used whenever the rule is used. The value of an attribute of a function can also be an attribute of a case. The value of the cases will not be stored in the rule, but the name of the attribute will be stored. The function will use the value of the specified attribute of the case being evaluated. Nominal diameter is an attribute of the function. The value of this attribute should be the value of the diameter of the case. The list of the combo box contains the attributes of the case. Select Then, "_ Diameter _" would have to be selected from the list.

![Function List](image)

The next attribute of the function is ‘Lower Limit’. The lower and upper limit values of the hole for the class RC1 are set to 0 and 0.392, respectively. The value of the next attribute, the real value of the hole of the case 'Real Value', should be retrieved from the case by selecting the second entry '_ Hole_' from the list. The first condition for the hole has been added to the rule.

The condition for the shaft needs to be added for the rule to be complete. The lower limit of the shaft for RC1 is -0.588. The upper limit is -0.308. Adding the ‘Non-locational’ condition for the function ‘A = B’ requires that the attribute ‘Locational’ is assigned the value ‘No’. To finish creating the rule, the condition: “Fit designation: RC1" is designated. Having validated this case, any situation that belongs to the RC1 class of fit will return the conclusion “Fit designation: RC1". Similarly, the rest of the rules are added.

5.5 Using the NRDR Tree

When the expert has defined all the rules, the fully populated tree is, therefore, saved and will become available for later loading and viewing. Upon viewing, the tree will appear as shown in Figure 8 complete with the condition(s) of each rule, the cornerstone case, and the scope of the rule.
6. SUMMARY

Proliferation in digital transfer of files representing mechanical databases requires that dimensions are properly tolerated. Consequently, AI content of mechanical CAD may have to become fully automated to where tolerancing of dimensions is executed during the actual design process and not at a later stage. In this work, we demonstrated that one can efficiently build an effective intelligent system to incrementally capture expert designer’s prescription in dimensional tolerancing. This was accomplished by utilizing a knowledge base system built on the Nested Ripple down Rules (NRDR) method. This intelligent system was successfully demonstrated in this paper by automating the process of tolerancing nominal dimensions based on the classical mechanical fit problem between a shaft and a hole. The system is able to perform both forward and backward fit schemes for cases like the one presented above. A forward scheme means that given a class of fit, i.e. LC1, LC2, etc., the software will return the upper and lower tolerance bounds for the nominal shaft/hole diameter of interest. A backward scheme means that given actual hole/shaft diameters the software would correctly identify the relevant class of fit.

Such a dimensional tolerancing knowledge management system may be integrated into smart CAD platform (see Figure 2) to help mechanical designers become more effective by automating the task of dimensional tolerancing of their designs in the future. Benefits of utilizing a smart system such as DesignAssistant include avoiding potential design conflicts and interfenses early in the development process thus reducing downstream errors, and engineering change orders (ECO’s), and reduced product lead times, improving quality, shortened product development process cycle, and increasing product performance-to-cost ratio.

7. ACKNOWLEDGMENTS

The authors wish to acknowledge the financial support of the University Research Board (URB) of the American University of Beirut. Also, the second author wishes to acknowledge the support of Consolidated Contractors Company through the CCC Doctoral Fellowship in Manufacturing.

REFERENCES