International Journal of Industrial Engineering Computations 3 (2012) 81–92

Contents lists available at GrowingScience

International Journal of Industrial Engineering Computations

homepage: www.GrowingScience.com/ijiec

A scheme for functional tolerancing: A product family in 3D CAD system

Haoyu Wang* and **Ravindra Thamma**

Central Connecticut State University, New Britain, CT 06050 USA

A R T I C L E I N F O

Article history:
Received 1 August 2011
Available online
10 August 2011

Keywords:
Tolerancing
Product family
Graph
CAD

A B S T R A C T

To meet the need for product variety, many companies are shifting from a mass-production mode to mass customization, which demands quick response to the needs of individual customers with high quality and low costs. The multifunctional nature of mechanical components necessitates that a designer redesign them each time when a component's function changes. The functional Geometric Dimensioning & Tolerancing (GD&T) specification, also called functional tolerancing, must be updated for each component. Currently, this is done by humans, and thus can be very time-consuming and error-prone. Functional tolerancing is one of the main obstacles to practical mechanical product family modeling. In this paper, a graph-based functional tolerancing scheme in 3D CAD is proposed. In the scheme, a product is generated by applying production rules to the graph of the base product, following customers' or manufacturing engineers' requirements. Functional tolerancing of each component of a product in the family is formulated as a non-linear constrained optimization (or cost minimization) process. Certain critical aspects of the scheme have been implemented in SolidWorks®, by using its Application Programming Interface (API) and C++. LEDA® and MATLAB® have been used to solve the graph and optimization problems.

1. Introduction

Traditional production systems, such as produce-to-order or assembly-to-order, work well with a low number of variants, but not when customers require a large product variety. Developing product families with design-to-order strategy has been recognized as means to support product variety with low costs and a minimal data redundancy. In many products, building block design or modular design is used successfully. Based on modular product architecture, product variety can be fulfilled through various combinations of modules. While the modular design method has been successfully applied in Very Large Scale Integration (VLSI) system design and personal computer design, it is hard to use in mechanical product design. The reason is that design economy dominates mechanical design (if one element were selected for each identified function, such systems would inevitably be too big, too heavy, or too wasteful of energy (Whitney, 1996)). The multifunction nature of mechanical components calls designers into redesigning them each time when their functions change. For a

* Corresponding author Tel.: +1-860-832-1824; fax: +1-860-832-1806
E-mail: wanghao@ccsu.edu (H. Wang)

doi: 10.5267/j.ijiec.2011.08.019

mechanical product, about 75% of the final cost is determined in the product design stage. Therefore, design tools are highly demanded to help designers face the challenge of mass customization. Currently, there is no well-defined design process for developing a family of mechanical products, nor is there research work to support designers when a variant product in the same product family is generated. In addition, when detailed design of a component is finished, there is no support for designers to specify geometric tolerances.

Tolerance specification is the topic that has been least studied so far. It usually involves a series of activities, such as identifying features to be toleranced and the required datum features, determining types of tolerances needed and material conditions, and finally, assigning some of the tolerance values as per functional requirement. Other tolerance values should be generated in the tolerance synthesis process. Traditionally, these activities heavily rely on the designer's experience, the empirical data, and/or the handbooks for designers and machinists. A systematic method is needed to automate this whole procedure, preferable in a CAD environment, incorporating domain-specific knowledge.

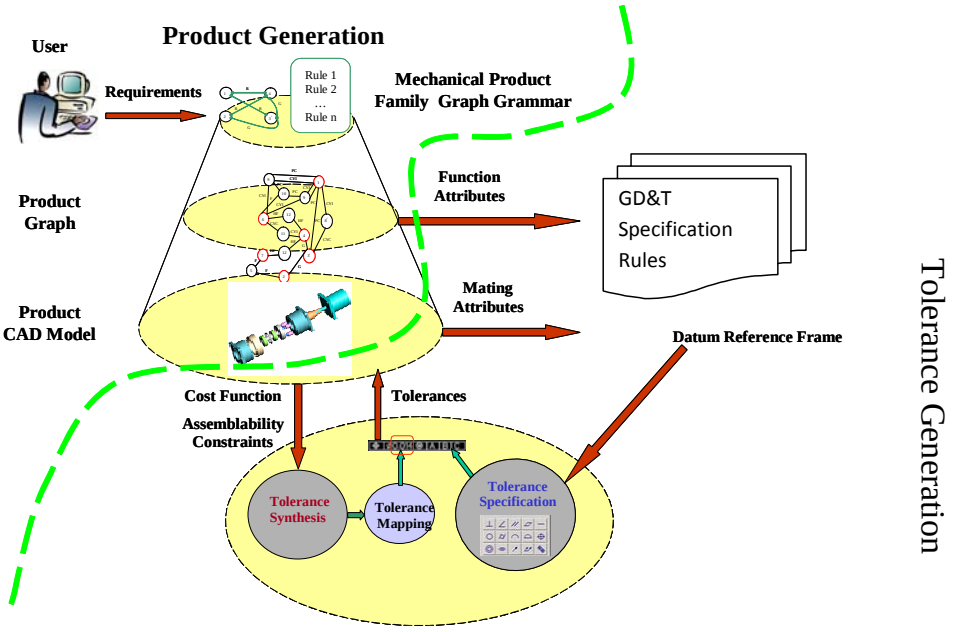

Fig. 1. Map of the scheme

In this paper, a graph grammar-based mechanical assembly family model is introduced. The generation of an assembly in the same family is modeled as the manipulation to the graph that represents the base assembly by applying graph production rules. The generated assembly variant is a graph with components as nodes and joints between components as edges. The joint information between components as well as the feature information of each component can help the designer in component design and tolerance specification. Fig. 1 illustrates the map of the overall scheme of the research in this paper. The research of this paper can be separated into two parts: mechanical product variant generation (product design) and tolerance generation (tolerance design).

In mechanical product variant generation, the user (a customer or a manufacturing engineer) enters his/her selections from a list of predefined requirements. The requirement selections are then mapped to the application conditions of a set of production rules. The production rules whose application conditions are satisfied are fired. The base product, which is represented by the graph of the product's mechanism, is manipulated by the fired production rules. The customized product variant of the mechanical product family, which is also represented as a graph, is then generated after all the fired production rules are applied.

The graphs that represent the base product and product variants are attributed graphs. The attributes of nodes and edges of the graph are to carry quantitative or qualitative design information. The information is utilized in tolerance generation for tolerancing of each component in the product. The GD&T specification rules are followed to generate Datum Reference Frames (DRFs) on each component. Other features on the component are toleranced to the DRFs, and the tolerance types and material conditions are generated based on the attributes of the features. Tolerance values or ranges of tolerance values are obtained through tolerance synthesis and tolerance mapping.

2. Review

In the past two to three decades, design methods were continuously being developed, tested, implemented in industry, and taught to the engineering community. Customer needs are first transformed to a repeatable functional representation, then to layouts and solution pieces, then to broad combinations and alternative products, and finally to an embodied realization that we can produce for the customer.

The need in the market for product variety requires formal design process to develop a family of products instead of single products. Characteristics of a product family range from flexible modular designs to robust and scalable designs, to standardized and flexible products. Martin and Ishii (1996) identified commonality, modularity, and standardization; Rothwell and Gardiner (1990) emphasized robust design; Wheelwright and Clark (1992) suggested designing “platform projects” that were capable of meeting the needs of a core group of customers but were easily modified into derivatives through addition, substitution, and removal of features. MacDuffie et al. (1996) looked at how the variety affected manufacturing within the automotive industry by studying empirical data.

Geometry and tolerance requirement of a specific mechanical component may change from one product to another in the same product family. A well-defined mechanical product family model should provide the logical relationships between components. These relationships are very important for component designing and tolerancing. However, none of the methods mentioned above has taken this issue into account. In this work, a graph grammar–based mechanical product family modeling method is presented to generate variants in a product family with the joints between components updated for tolerance specification.

Tolerance specification usually encompasses a series of activities, such as the identification of features to be toleranced and their required datum features, the determination of the types of tolerances needed. As mentioned earlier, tolerance specification is the topic that has been least explored so far. In practice, the tolerances are specifications by the designer, mainly based on experience and/or empirical information. The concept of topologically and technologically related surfaces (TTRS) (Clement, 1996) is used in specifying tolerances on components. Tolerance types are selected based on the geometric relations between the functional features. But the selection of datum features, datum precedence, and selection of material condition are missed in this method. Linares (2002) presented a tolerance specification method by introducing a concept called Functional Group, but very limited cases of tolerance specification are covered. A tolerance specification method based on the influence of contacts is presented by Anselmetti (2002). Kandikjan (2001) proposed a tolerance advisor for tolerance specification. The scheme is totally based on the component, not the assembly. In this paper, the mirror method is proposed for an assembly–oriented functional tolerance specification to cover selection of datum features, datum precedence, selection of material condition, and selection of geometric tolerance types by using information stored in the generated products’ graph model.

3. Graph-based Representation of an Assembly

Figure 2 shows an example of the graph formalism of the graph of a piston assembly, where the bigger circles represent nodes (components), the smaller circles represent ports (features) of the nodes, and the lines between ports are edges (joints). Labels and attributes of nodes, edges, and ports are also illustrated.

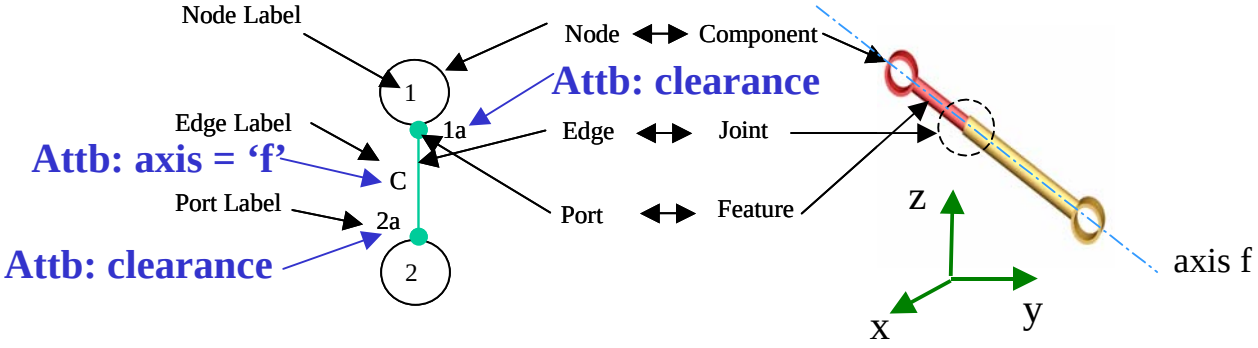

Fig. 2. Graph representation of a piston assembly

3.1 Graph grammar of a mechanical product family

The term "graph grammar" generally means a method for generating a set of graphs from a starting graph. Manipulations on the starting graph are carried out by applying production rules. All graphs that can be derived by applying production rules to a starting graph construct the language of this graph grammar. This research adopts graph grammars as tools to model a mechanical product family. The strategy of graph grammar–based mechanical product family modeling is to design graph grammars to represent the organization of mechanical product family elements and accordingly to transform the variant derivation process into a process of assembly graph derivation.

Graph grammar with ordered production rules is called programmed graph grammar. In a programmed graph grammar, the sequence of executing a set of productions can be expressed in a control diagram. Product variants of the family can be derived by applying production rules according to the control diagram to modify the starting graph which represents the base product. The resultant graphs are the graph models of desired variants.

A Programmed Attributed Graph Grammar (PAGG) is defined as a nine-tuple:

$$GG = (V, W, X, A_V, A_W, A_X, S, P, CD)$$

where $V = \{C_i\}$ is a set, consisting of node labels (i.e., names or IDs) of all components in the product family; $W = \{C_i \times C_j\}$ is a set, consisting of edge labels that indicate the joints between the components; X is a set, consisting of port labels that represent features; A_V is a set, consisting of node attributes representing attributes of components; A_W is a set, including edge attributes representing attributes of joints; A_X is a set, consisting of port attributes; S is the starting graph representing the base product; $P = \{p_i\}$ is a set, including all production rules defined for graph manipulation to generate variants; and CD is the control diagram over P, specifying the order by which productions are applied so that the variants can be derived.

All attributed graphs that can be derived by graph grammar GG as defined above are termed language of the graph grammar. The derivation steps are: 1) starting with a starting graph S; 2) applying all applicable productions P in an order specified by the control diagram CD.

Set *V* in *GG* represents component names in a mechanical product family. It changes from family to family. Hence, it is trivial to list all possible component names. Component attributes A_V may include parameters such as list of features, material, etc.

1) The Base Product

The base product should include common components or core components that all products in the same family contain. Other components in the same product family are termed optional components. The core components together with joints between them should fulfill common functions of the products in the family. Graph representations of a range of planar linkages, planar geared mechanisms, planar cam mechanisms, spherical mechanisms, and spatial mechanisms have been developed and cataloged in a graph atlas (Tsai 2001). The functional scheme and its graph representation of a crank-slider mechanism are shown in Figure 3. The circle enclosing the node indicates that the node is "grounded" or "fixed."

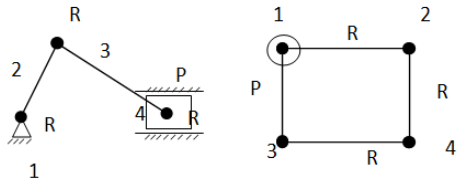

Fig. 3. Crank-Slider mechanism and its graph representation

2) Production Rules

A production rule P has two parts: the operation (*O*) and its application conditions (π). The operation $O = (g_l, g_r, T, P)$ designates how the left-hand side (LHS) graph, g_l, is replaced by the right-hand side (RHS) graph, g_r, with respect to embedding transformation, denoted by *T*, or port transformation, denoted by *P*.

Addition

A particular component carrying out certain additional functions can be added to a base product to create a new product variant. Adding a component or subassembly *C* to a base product *BP* is equivalent to adding the graph *g(C)* to the host graph *g(BP)*. Addition operation can be represented by a four-tuple: $Addition = (g_l, g_r, T, P)$:

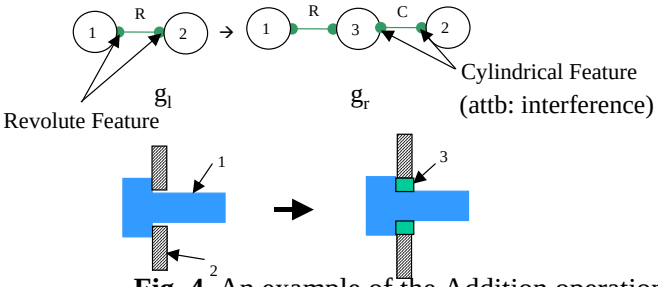

Fig. 4. An example of the Addition operation

where g_l is the conditional graph which implies that the base product provides interface for component *C* to be added; g_r is the resultant graph after component *C* being added; and *T* is the embedding transformation function that specifies that all the edges connected to the components in g_l will be connected to the corresponding components in g_r.

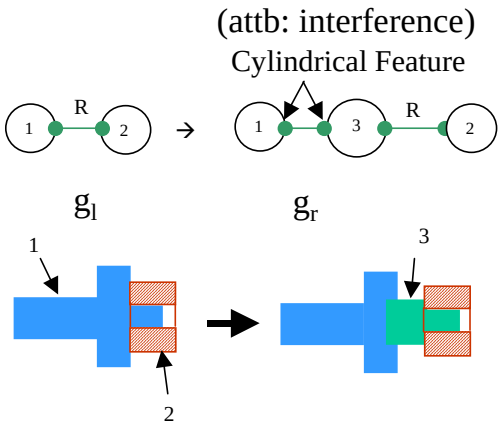

Fig. 5. An example of the decomposition operation

Decomposition

This category of operations serves the purpose of resembling the subsequent splitting of components into components that have simpler shapes. This kind of operation may be required by manufacturing or assembly practices. This category shows similar expressions to those in the "addition" category, with the slight difference that the created nodes here only serve part of the functions (joints with other components) of the original components. Decomposition operation can be represented by a four-tuple: *Decomposition* = (g_l, g_r, T, P), similar to the "addition" operation. An example of the Decomposition operation is shown in Fig. 5. In the example, component 1 is decomposed into the modified component 1 and component 3, so that the cylindrical feature mating with the gear (component 2) is transformed from the original component 1 to the component 3. This might happen when a component with many functional features is easier or more economical to be manufactured if it can be decomposed into several components.

Modification

Operations belonging to this category are intended not to extend the graph model obtained so far by adding nodes but rather to perform necessary modifications concerning the ports and attributes of nodes or relations among nodes. This may happen when we want to replace a component with one that has similar functions. For example, a customer may want to change a gear with one gear feature into a gear with two gear

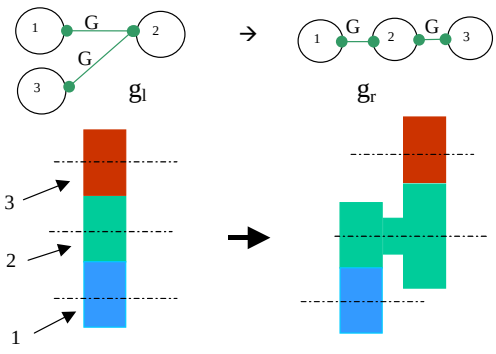

Fig. 6. An example of the Modification operation

features to get a larger input/output ratio. This is shown in Figure 6. Modification operation also can be represented by a four-tuple as above: *Modification* = (g_l, g_r, T, P).

After defining the operations used to modify graphs for variety generation, conditions under which the operations can be performed have to be specified, that is, when components should be added, decomposed, or modified. Application conditions are introduced for this purpose. When application conditions are satisfied, the applicability predicate, π, is TRUE. Otherwise, π is FALSE. Application conditions can be expressed as functions of customers' or manufacturing engineers' selected requirements or requirement values. A production rule is defined as a two-tuple: $P = (O, \pi)$, where 1) $O \in \{Addition, Decomposition, Modification\}$ is the operation; 2) $\pi \in \{TRUE, FALSE\}$ is the applicability predicate, which is a logic function of application conditions. A few examples of production rules are given as follows, where R_i indicates requirement i of the product.

$P1 = (O=Addition\ (Bearing), AC = (\alpha(R1) \in \{TRUE\}))$

$P2 = (O=Decomposition\ (Holder), AC = (\alpha(R3) \in \{step\}))$

where α is a function to get the value of the requirement Ri and λ is a function to get the name of the requirement Ri.

$\lambda\ (R1) = Minimize\ Friction, \alpha(R1) \in \{TRUE, FALSE\}$

$\lambda\ (R3) = Ease\ Manufacturing, \alpha(R3) \in \{step, non\text{-}step\}$

3) Graph Derivation

Deriving a product variant may involve more than one step of modification of the base product. The process of modifying a base product to a customized one can be modeled as a series of graph derivations by executing certain production rules. The derivation of a graph, g', from a graph, g, by means of a production, p, follows the following procedures:

1. Check whether g_l is a subgraph in g, and check if the application condition is true. If both condition s are fulfilled, continue to step 2.

2. Substitute g_l, including all incoming and outgoing edges, with the nodes and edges of g_r.

3. Transform the embedding of g_l in g into g_r in g'.

4. Update the ports of nodes in g_r.

4) Control Diagram

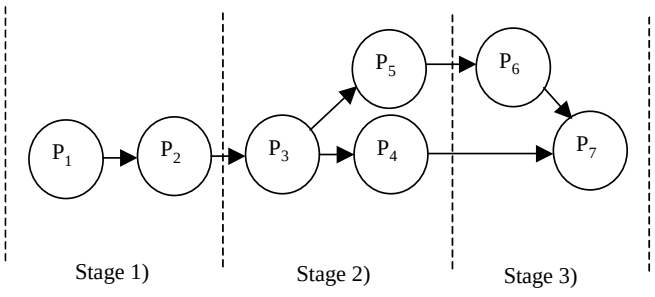

Fig. 7. An example of a control diagram

Usually, a number of productions will be involved in the process of graph derivation. The order for a collection of productions to be invoked is specified by the control diagram. The control diagram in the graph grammar of a product family expresses the order in which productions defined for this product family are to be executed. Our method is to go through this process by imitating a real designer's design practice in three stages: 1) adding new components or modifying existing

components in the mechanism to fulfill a customer's or an engineer's functional requirements; 2) increasing performance of the product by adding components, such as gaskets or bearings; and 3) adding positioning or fastening components to secure the relative positioning of the components in the assembly.

To map customers' or manufacturing engineers' requirements to a variant design, the checking of application conditions should start from the production that is at the beginning position of the control diagram. The path in the control diagram with the maximum number of applicable productions (application condition is TRUE) is chosen and all feasible productions along the path will be applied. A variant is generated when no more production rules in the path of the control diagram can be applied. Figure 7 shows an example of a control diagram. In figure 8, a graph that represents the base product of a planetary gear train is transformed into a graph that represents one possible product in the planetary gear train product family. Figure 9 shows the product's physical model in SolidWorks®. For details of the graph generation process, please refer to Wang, 2005.

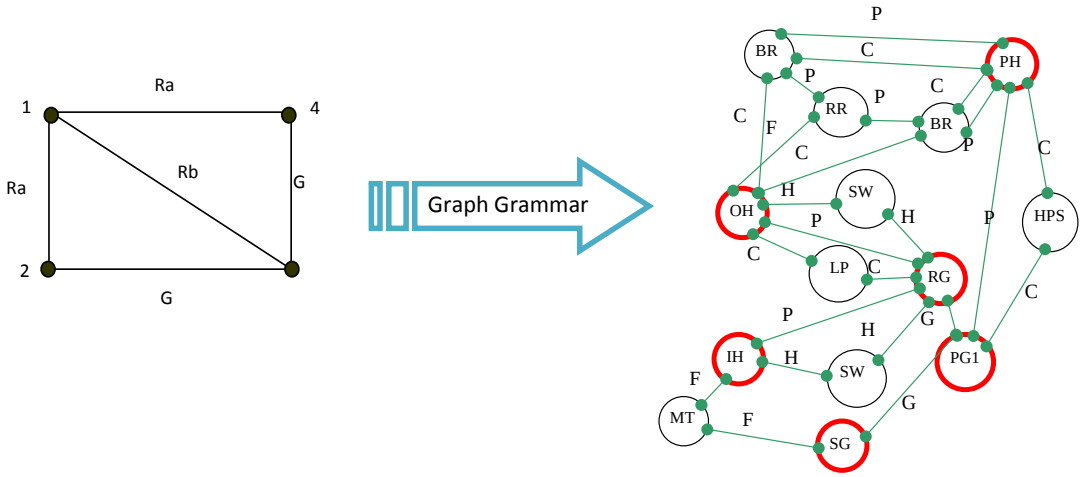

Fig. 8. Example of graph transformation

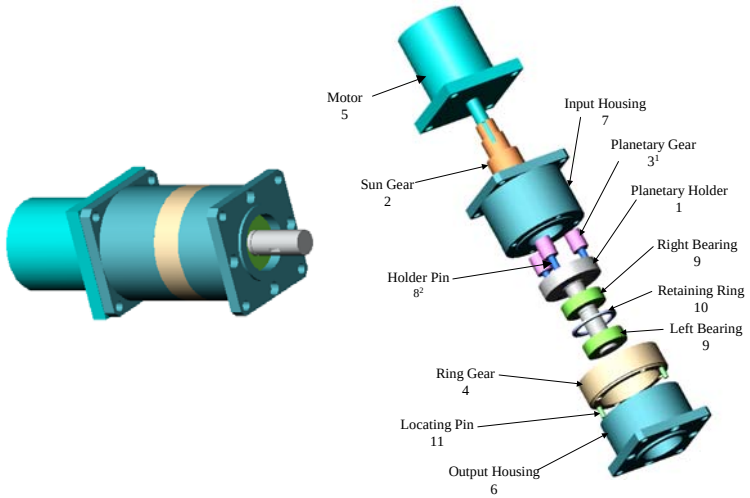

Fig. 9. CAD model of the planetary gearbox for the graph in Fig. 8

3.2 Functional tolerancing in 3D CAD

With enormous customer demands, commercial CAD systems open many more spaces for third-party developers to access the CAD model through API (Application Programming Interface) and develop

software for their own use. In-house software can therefore be quickly prototyped and tested. SolidWorks has been chosen as the CAD platform to implement the proposed tolerance specification and tolerance synthesis methods owning to the following reasons:

- SolidWorks is a very popular CAD system based on a solid model which is 3D in nature. This matches our purpose for 3D tolerance specification and synthesis.
- The API is well documented.
- The API is easy to use and sample codes are available.
- There is noo extra charge for the API.

The tolerance specification module and the tolerance synthesis module are developed as an add-in DLL (Dynamic Link Library) in SolidWorks by using the Object-Oriented Design method through SolidWorks' API and MS Visual C++. The system structure is in Figure 10.

The user interacts with the tolerance specification module and tolerance synthesis module through SolidWorks Graphical User Interface (GUI) and GUI of the two modules by Microsoft Foundation Class (MFC).

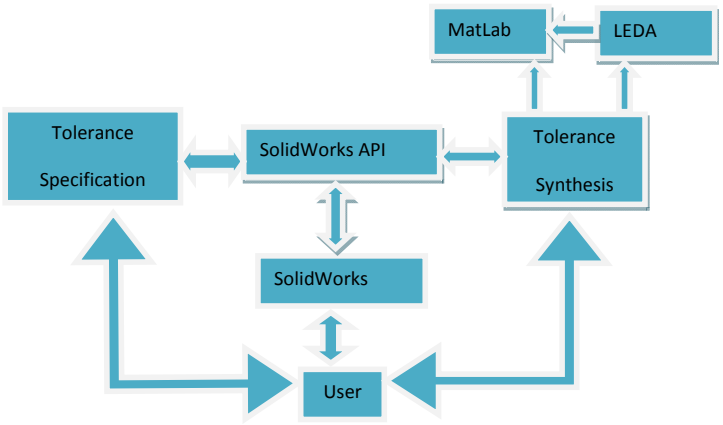

Fig. 10. The structure of the implementation

3.3 Example

1)Tolerance Specification

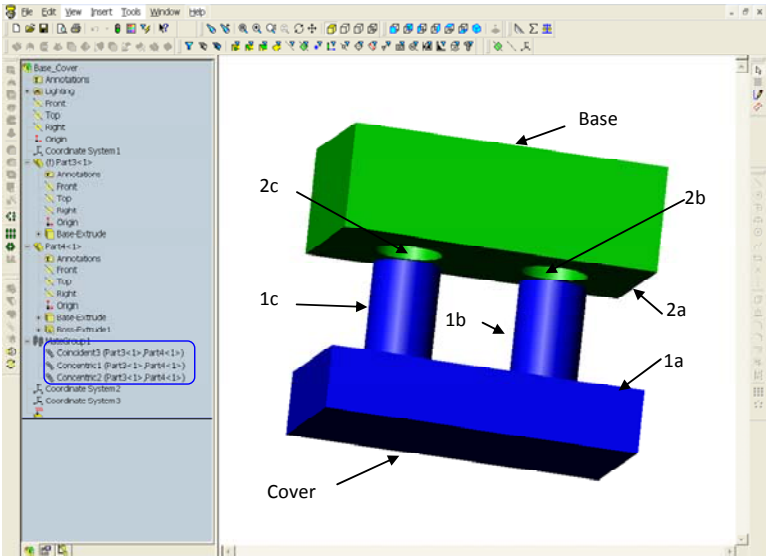

Fig. 11. The base-cover assembly in SolidWorks

A base cover assembly is used to illustrate how the tolerance specification module and the tolerance synthesis module work in SolidWorks. The solid model of the assembly is shown in Figure 11. As can be seen, there are three mates between the Cover and the Base: a coincident mate between 1a and

2a, a concentric mate between 1b and 2b, and a concentric mate between 1c and 2c. The tolerance specification module and tolerance synthesis module are programmed as an add-in to SolidWorks in Dynamic Link Library (DLL) file format. This add-in file can be loaded by opening the file directly in SolidWorks or by checking the respective check box in the Add-ins dialog box.

Tolerance Synthesis

In this paper, only features in mates are taken into account for tolerance synthesis to simplify the problem. The tolerance synthesis module shows mates between components and the connection graph as shown in Fig. 12. As shown in Fig. 13, three points are selected on each component for mating features. (37.5,20,0) is for T1a, T2a, and G1a/2a; (63, 20, 12.5) is for T1b, T2b, and G1b/2b; and (12, 20, 12.5) is for T1c, T2c, and G1c/2c. (0,0,0) is the origin of the base component (part 2) and is the point for T1; (0,25,0) is the origin of the cover component (part 2) and is the point for T2.

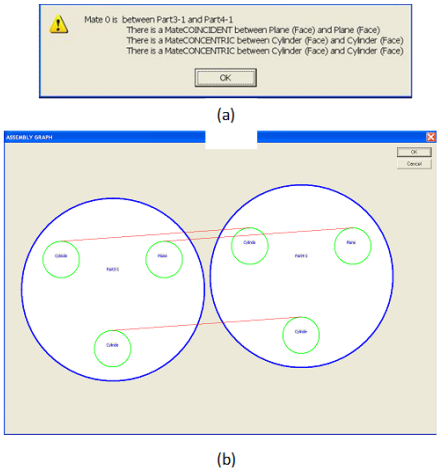

Fig. 12. Mates between components and the assembly graph

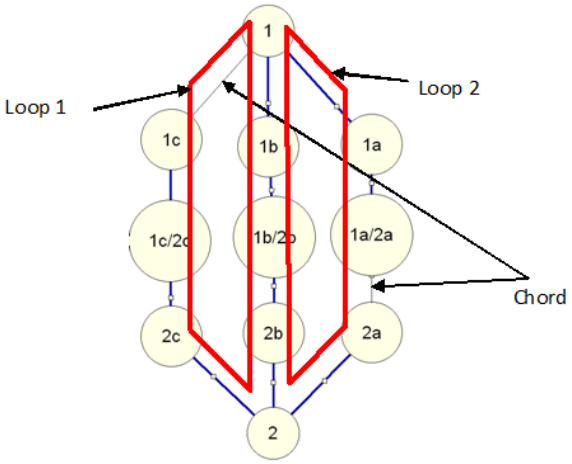

Fig. 13. Spanning tree of the modified connection graph

Table 1-1 shows the labels of the nodes of torsors, their points' coordinates, and LCSs. By using LEDA's graph algorithm we get the spanning tree, as shown in Figure 13. Therefore, we know that there are only two independent loops: (Loop 1)1→1b→2b→2→2c→1c→1 and (Loop 2) 1→1a→2a→2→2b→1b→1. The two loops are represented by two rows of sequenced integers (0, 1, or –1) as shown in Table 1-2.

Table 1-1
Torsors' labels, coordinates, and LCSs of the Cover-Base assembly

Node Label	Torsor	Coordinates (x,y,z)	LCS (X-axis;Y-axis;Z-axis)
0	T_2	(0,0,0)	(1,0,0;0,1,0;0,0,1)
1	T_{2a}	(37.5,20,0)	(1,0,0;0,-1,0;0,0,-1)
2	T_{2c}	(12,20,12.5)	(1,0,0;0,-1,0;0,0,-1)
3	T_{2b}	(0,25,0)	(-1,0,0;0,1,0;0,0,-1)
4	T_1	(63,20,12.5)	(1,0,0;0,1,0;0,0,1)
5	T_{1a}	(37.5,20,0)	(1,0,0;0,1,0;0,0,1)
6	T_{1c}	(12,20,12.5)	(-1,0,0;0,-1,0;0,0,1)
7	T_{1b}	(63,20,12.5)	(1,0,0;0,1,0;0,0,1)
8	$G_{1a/2a}$	(37.5,20,0)	(1,0,0;0,1,0;0,0,1)
9	$G_{1c/2c}$	(12,20,12.5)	(-1,0,0;0,-1,0;0,0,1)
10	$G_{1b/2b}$	(63,20,12.5)	(1,0,0;0,1,0;0,0,1)

Table 1-2
Loop Matrix (note: N means node, L means loop)

					N						
L	0	1	2	3	4	5	6	7	8	9	10
1	-1	0	1	-1	1	0	-1	1	0	-1	1
2	-1	-1	0	1	1	1	0	-1	1	0	-1

To solve the optimization (minimization) problem as formulated in this paper, the "fmincon" nonlinear constrained optimization solver in MATLAB is used. Results of the optimization run are shown in Table 1-3. The optimal deviation parameters are used for computing the tolerance values/ranges (Table 1-4).

Table 1-3
Optimal deviation parameters of torsors

Node	dx	dy	dz	Rx	Ry	rz
0	0.0099	0.01	0.0099	0.0017	0.0063	0.0005
1	-0.0001	0	0.01	0.01	0.01	0.0024
2	0.01	0.01	-0.0001	0.01	0.01	0.0011
3	0.01	0.01	0	0.01	-0.0028	0.0018
4	0.01	0.0099	0.01	0.0017	-0.0018	0.0015
5	0.0001	0	0.01	0.01	0.01	-0.0017
6	0.01	0.01	-0.0001	0.01	0.01	0
7	0.01	0.01	0	0.01	0.01	-0.0008
8	0.0001	0	0	0.01	-0.0028	-0.0017
9	0.01	0.01	-0.0001	-0.01	-0.0012	0.0001
10	0.0099	0.01	0	-0.01	-0.0036	-0.0008

Table 1-4
Results of tolerance values or ranges

Feature	Tolerance Type	Deviation Mapping	
1a	Size	TL = 0.165	TU=0.585
2a	Size	TL = 0.165	TU=0.585
1b	Perpendicularity	f_{min}= 0.1773	f_{max}= 0.1773
2b	Perpendicularity	f_{min}= 0.1240	f_{max}= 0.1375
1c	Position	f_{min}= 0.1773	f_{max}= 0.1773
2c	Position	f_{min}= 0.1626	f_{max}= 0.1909

Table 1-4 listed the recommended ranges for size tolerances. But for perpendicularity and position tolerances, f_{max} and f_{min} values give the possible ranges of the values *TU*, *TL*, and *Tp* constrained by the mapping relations. These inequalities are planes in the *TU*, *TL*, *Tp* space and all such inequalities for each feature define the tolerance zone (Wang, 2006).

4. Conclusions and future work

A graph grammar-based mechanical product modeling scheme has been proposed. A product is generated by applying production rules to the graph of the base product, the mechanism. Both base product and end product are represented by graphs with components as nodes and joints between components as edges. The generated attributed graph is a data structure which represents the joint and feature information of the customized product. In the end product, the joints between a component and other components in the product represent the functions concerning how the components are to be related to each other geometrically. Therefore, the joints information can be used in designing and tolerancing. Further efforts should be made to extend research in the following areas. The users of the scheme of mechanical product family modeling can be roughly classified into "designers" and "experts." While designers only have knowledge of designing the product, experts know how to

design the product as well as how to translate the design knowledge to production rules for the product family. As described in section three, the key elements of a graph grammar of a mechanical product family are: a starting graph which represents the mechanism of the product, production rules, and a control diagram. Developing the three elements might not be a problem for an expert (even if it might take him or her months), but it could be very challenging for a designer who has very limited knowledge about graphs, production rules, etc. In order to make the proposed scheme convenient and efficient for all kinds of users, a generic system for mechanical product family modeling should be developed. It should allow users to select and edit starting graphs. Production rules should be generated based on user input such as customers' or manufacturing engineers' requirements, changes of features on components due to said requirements, etc. Users should be able to generate or edit the control diagram by their knowledge of design sequence and the support of the system.

There are two interesting questions about production rules that might indicate the directions of future study. One is "How do we know the production rules of a mechanical product family are complete?" The other is "How do we know if the production rules are optimal?" A complete production rule set should be able to generate product variants that meet all possible combinations of requirements for a product family. It should also represent all possible changes of the product configuration due to one specific requirement. An optimal production rule set is a complete one that has the minimum number of production rules. Future research about criterias that help users in evaluating the completeness of the production rules is needed to help searching for the optimal product design.

References

Anselmetti, B., Thiebaut, F., & Mawussi, B. (2002). Functional Tolerancing Based on the Influence of Contacts. *APSE 2002 Summer Topical Meeting on Tolerance Modeling and Analysis*. Charlotte, North Carolina. CD ROM. 15-16 July 2002.

Clement, A., Valade C., & Rivière A. (1996). The TTRSs: 13 oriented constraints for dimensioning, tolerancing and inspection. *Proceedings of the Advanced Mathematical Tools in Metrology III Conference*. Berlin. September, 25-28.

Kandikjan, T., Shah, J.J., & Davidson, J.K. (2001). A mechanism for validating dimensioning and tolerancing schemes in CAD systems. *Computer-Aided Design*, 33, 721-737.

Linares, J. M. (2002). Synthesis of tolerancing by functional group. *Journal of Manufacturing Systems*, 21(4), 260-275.

MacDuffie, J. P., Sethuraman, K., & Fisher, M. L. (1996). *Product Variety and Manufacturing Performance: Evidence from the International Automotive Assembly Plant Study*. Management Science, 42, 350-369.

Martin, M. V., & Ishii, K. (1996). Design For Variety: A Methodology for understanding the Costs of product proliferation. *ASME Design Engineering Technical Conferences*, Irvine, CA. 96-DETC/DTM-1610.

Rothwell, R., & Gardiner, P. (1990). *Robustness and Product Design Families, Design management: a handbook of issues and methods*. Moakley. Cambridge, MA. Basil Blackwell Inc.

Wang, H., & Roy, U. (2005). A Graph-Based Method For Mechanical Product Family Modeling And Functional Tolerancing. *The 31st Design Automation Conference, ASME IDETC/CIE*.

Wang, H., Pramanik, N., Roy, U., Sudarsan, R., Sriram, R.D., Lyons, K.W. (2006). A Scheme for Mapping of Tolerance Specifications to Generalized Deviation Space for Use in Tolerance Synthesis and Analysis. *IEEE Transactions on Automation Science and Engineering (T-ASE).* 3(1), 81-91.

Wheelwright, S. C., & Clark, K. B. (1992). *Revolutionizing Product Development: Quantum Leaps in Speed, Efficiency and Quality*. New York, The Free Press.

Whitney, D. (1996). Why mechanical design cannot be like VLSI design. *Research in Engineering Design*, 8, 125-138.