Available online at www.sciencedirect.com

ScienceDirect

Procedia CIRP 44 (2016) 55 – 60

www.elsevier.com/locate/procedia

6th CIRP Conference on Assembly Technologies and Systems (CATS)

Optimum Overall Product Modularity

Mohamed Kashkoush and Hoda ElMaraghy*

Intelligent Manufacturing Systems (IMS) Centre, University of Windsor, 401 unset Avenue, Windsor, ON, N9B 3P4 Canada

* Corresponding author. *E-mail address:* hae@uwindsor.ca

Abstract

Modularity in product architecture is beneficial to both product development and manufacturing. Several methods exist for clustering product components into modules all of which, with few exceptions, do not consider the hierarchical structure of the product. Products architecture consists of a number of hierarchical levels, which add a useful dimension to modularity analysis. Designing products architecture that maximizes modularity over all levels of the product structure (i.e. overall modularity) is the main objective of this work. Interactions between various product components are represented using a Design Structure Matrix (DSM). The product architecture is represented by product structure tree in the form of a binary rooted tree. A novel Mathematical Programming Model is developed to construct the corresponding product structure tree for a given product which ensures optimal modularity at all hierarchical levels, without prior knowledge of their number. The proposed optimal modular product architecture design method is demonstrated using a real product case study. Optimal overall modularity leads to better management of product changes and variety and more cost-effective product development and manufacturing.

Peer-review under responsibility of the organizing committee of the 6th CIRP Conference on Assembly Technologies and Systems (CATS)

Keywords: Modularity; Product Architecture; Granularity; Design Structure Matrix; Integer Programming

1. Motivation

Constructing the product architecture is a critical design activity that affects all subsequent product development steps such as analysis, modeling and prototyping, manufacturing, assembly and supply chain. A widely accepted definition of product architecture is: “the organization of the functional components of a product into physical units and the interaction among them” [1]. Products with integral architecture are constructed with a combination of the lowest level (individual) components without intermediate sub-assemblies; while products with modular architecture are built of one or more levels of sub-assemblies [2]. A module in a modular product refers to a structurally independent element or a sub-assembly of a larger sub-assembly with clearly defined interfaces [3, 4]. Modularity of product architecture is not an absolute property; in fact, it is a relative property that can be quantitatively assessed and analyzed.

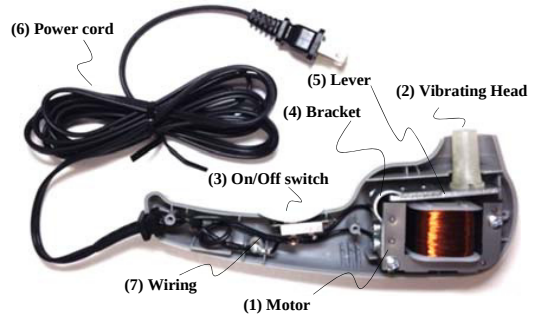
Modular product architecture has become an important product development topic in the last few decades [5]. It allows manufacturers to cost-effectively handle and develop complex products by decomposing them into simpler sub-units or modules [6]. The benefits of modularity regarding product functionality, product design, maintenance, services, testing and verification, production, supply chain, and other activities have been discussed by many researchers in various fields [5, 7, 8]. A comprehensive discussion of benefits of modularity is provided by Ulrich [1]. They include: (1) efficient management of product changes and updates as manufacturers can easily change some functions of subsequent product generations by simply changing some modules, (2) better product variety management as variety could be simply created by having various combinations of modules, and (3) enhanced product development process since design tasks can be properly decoupled and carried out concurrently.

2. Literature Review

Several methods exist for designing modular products [9] and can be classified into two main groups [10]: function-based and matrix-based methods. Function-based methods [11-13] identify modules by mapping the functional decomposition of a product to its physical architecture. These

Peer-review under responsibility of the organizing committee of the 6th CIRP Conference on Assembly Technologies and Systems (CATS)
doi:10.1016/j.procir.2016.01.023

methods are criticized for not being able to adequately address the interfaces between physical elements of the product [2]. Matrix-based methods [2, 14, 15], used in this research, identify modules by reconfiguring the product Design Structure Matrix (DSM) which represents the product architecture [3].



(a) Hand-held massager

		Motor	Vibrating Head	On/Off Switch	Bracket	Lever	Power Cord	Wiring
		1	2	3	4	5	6	7
Motor	1	1	0	0	1	0	0	1
Vibrating Head	2	0	1	0	0	1	0	0
On/Off Switch	3	0	0	1	0	0	1	1
Bracket	4	1	0	0	1	1	0	0
Lever	5	0	1	0	1	1	0	0
Power Cord	6	0	0	1	0	0	1	0
Wiring	7	1	0	1	0	0	0	1

(b) Massager DSM

Fig. 1. DSM developed for **Wahl 4120-600** hand-held body massager (http://massagers.wahl.com/)

The Design Structure Matrix (DSM), first introduced by Steward [16], is the most common tool used to model interaction between components of a given product [17] and is, therefore, used by many researchers to model and study product architecture [18]. Fig. 1 illustrates the DSM constructed for a hand-held body massager made of seven components in addition to the casing. Each row and column of the matrix represents one of the massagers' components. Cells filled with 1 in the matrix refer to the existence of interaction between the two components corresponding to that cell. The type of interaction modeled in this example is on/off spatial interaction (physical adjacency). Other types of interactions (e.g. energy and information) and different interaction coding schemes (e.g. quantified interaction) may also be used [19]. Product modules are identified by re-arranging the DSM into clusters along the matrix diagonal to maximize interaction within clusters and minimum interaction outside. Several methods have been used in literature to identify products modules by clustering the DSM based on different modularity measures.

Existing methods for designing modular product architecture provide a flat map for modules and their interaction without considering their hierarchical levels of sub-assemblies representing the structural arrangement of components within each module. The only exception to that finding is the work on granularity analysis of modular products and manufacturing systems architectures by AlGeddawy and ElMaraghy [20-22].

AlGeddawy and ElMaraghy [20] used cladistics analysis, which is a hierarchical classification method commonly used in Biology [23], in order to identify product modules based on DSM architectural representation. Along with the identified modules, the method further provides the hierarchical structure of the product in a binary rooted tree format (cladogram). The ultimate objective of that method is to find the optimal hierarchical level of the product architecture (granularity level) for product modularization. Although, the cladistics method has provided better modularity results compared to earlier research results in the subject; the main focus of the study was to find the optimal granularity level and not to optimize the product architectural tree. Finding the product structure tree (number of tree levels and clusters of components formed at each level) that leads to the optimal total modularity over all the tree levels was not an objective for the cladistics analysis-based method.

Inspired by the work AlGeddawy and ElMaraghy [20], this paper addresses product modularity from a new perspective to provide the product architecture, represented by the product structure tree in the form of binary rooted tree, for maximum overall modularity at all levels of the product structure. In this research, every potential module of a given product is viewed as if it was a product on its own the architecture of which needs to be designed for optimal modularity. This allows all modularity benefits discussed earlier to be propagated throughout the entire product at all its hierarchical levels.

3. Proposed Modular Product Architecture Design Methodology

Product architecture is modeled as shown in Fig. 2 using binary rooted trees (i.e. product structure tree or product tree for short). The tree in Fig. 2 represents the structure of a product with eight components. The product has four hierarchical levels and each level has its own modularity. For instance, at level 1 the product has only two modules; module 1 {6, 8, 5, 7, 3, 4} and module 2 {2, 1}. Whilst, at level 2, the product has three modules; module 1 {6, 8, 5, 7, 3}, module 2 {4} and module 3 {2, 1}. Modules # 1 and # 2 at level 2, when combined, they form module # 1 of level 1.

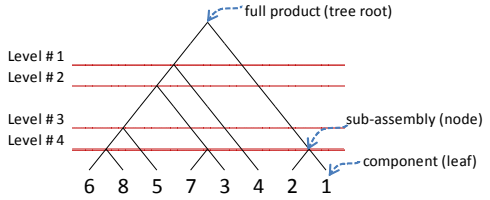


Fig. 2. Binary rooted tree representation of product structure

Therefore, each level of the tree provides more details related to components arrangements for modules at higher levels of the tree. Hence, the tree represents the components

arrangement (clusters) within each module starting with the level of individual components to the level at which the product is only one big module which includes all the components (i.e. full product). Fig. 2 is schematically represented in Fig. 3 to further illustrate the notion of having multiple levels of product architecture modularity.

The modularity is modeled and assessed at any level of the tree using the corresponding re-ordered DSM. The Modularity Index (MI) introduced by AlGeddawy and ElMaraghy [20], which is also equivalent to CE used by Pandremenos and Chryssolouris [24], is utilized in this research. The MI counts the total number of intra-relationships between different modules as well as missed inter-relationships between components of the same module. MI is given by Eq. 1, where I is the number of '1' DSM cells outside generated clusters, and Z is the number of '0' DSM cells inside those clusters. Improved clustering corresponds to smaller MI values.

$$MI = I + Z \quad (1)$$

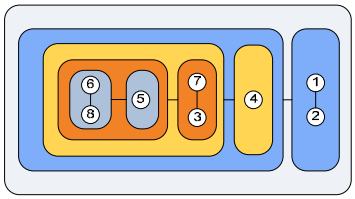


Fig. 3. Schematic representation of the product tree shown in Fig. 2

4. Non-Linearly Constrained Mathematical Programming Model

A non-linear integer programming model with non-linear objective function and constraints is first formulated then the non-linear objective function is linearized. The input to the model is the DSM of a given product. Only Binary (0-1) DSMs are considered. The output is an encoded matrix representing the optimal product structure tree (i.e. the tree corresponding to optimal overall modularity).

4.1. Tree-to-matrix encoding scheme

The proposed model handles product trees in the form of encoded matrices. A binary rooted tree representing the structure of *n* components has n-1 tree nodes. Hence, for a tree of *n-1* nodes a matrix of *n-1* rows and *n* columns is used to represent it (i.e. a matrix of size (*n-1*) x *n*). The matrix columns represent component numbers and its rows represent node numbers. Binary (0-1) values are used to indicate the presence or absence of a given component in any given node.

For instance, node number 2 in the tree shown in Fig. 4 has four components 1, 2, 3, and 4, thus, row number 2 in the corresponding encoded matrix will take the value 1 in cells number 1, 2, 3, and 4, while cell number 5 will have a 0 value due to the absence of component 5 in node 2. Therefore, any encoded matrix will always have one of its rows full of ones representing the tree root node, to which all components belong. Nodes numbering is arbitrary and accordingly many matrix representations are possible for the same tree. Therefore, if the rows of the matrix are altered to any other order, they still represent the same tree. This tree-to-matrix encoding scheme was developed by the authors to encode product assembly trees [25].

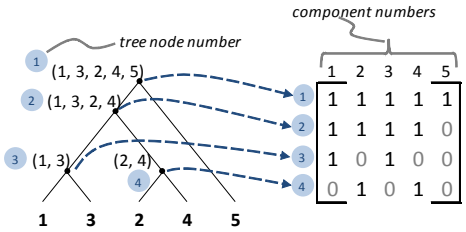


Fig. 4. Proposed tree-to-matrix encoding scheme (modified from [25])

4.2. Non-linear integer programming model formulation

The model input parameters are as follows:

n	Total number of components
m	Number of nodes of the hierarchical clustering tree (n-1)
DSM_{ij}	A binary (0-1) element in the i^{th} row and j^{th} column of the DSM matrix

The decision variables are:

c_{ui}	A binary (0-1) element in the u^{th} row and j^{th} column of the encoded matrix representing the product tree
x_{uv}	A binary (0-1) element valued at 1 if node v is directly branched from node u
y_{ui}	A binary (0-1) element valued at 1 if component i is directly branched from node u
f_{uv}	A binary (0-1) element valued at 1 if node v forms a module at level u of the product tree
h_{ui}	A binary (0-1) element valued at 1 if the individual component i forms a module at level u of the product tree
q_u	A binary (0-1) element valued at 1 if the u^{th} assumed tree level is to actually exist
l_u	The level of node u; given by no. of components that belong to node u

The proposed model formulation consists of seven sets (arrays or vectors) of variables as shown in the list of decision variables. The first three sets of variables (c, x, and y arrays) are used to define the product tree. The c array represents the encoded matrix for any given product tree (illustrated in Fig. 4), while the x and y arrays are both used to describe direct relationships between the nodes and leaves of the tree. The x and y arrays are the actual variables used in the model to control the structural feasibility of generated product trees. The c array, which is the ultimate outcome of the model, is the result of combining the two arrays which will be shown later in the description of the constraints.

The f and h arrays are the main sets of variables used to define nodes and components that form modules at each product tree level. The number of tree levels is not known beforehand, so it is initially assumed that the product tree is structured in a way in which a different level is to exist at each node of the tree (excluding the root node). The q vector identifies which of those assumed levels should exist according to the actual tree structure. The l vector is an auxiliary set of variables that define the tree level where each node lies, to be used by the model to check nodes clustering feasibility. Using the listed parameters and decision variables, a non-linear integer programming model is formulated with 12 sets of constraints as follows:

$Minimize$ OMI =

$$\sum_{u=1}^{m}\sum_{v=1}^{m}\sum_{i=1}^{n}\sum_{j=1|i<j}^{n} 2(1-DSM_{ij})(c_{vi}\,c_{vj})f_{uv} + \sum_{u=1}^{m}\sum_{v=1}^{m}\sum_{i=1}^{n}\sum_{j=1|i<j}^{n} 2DSM_{ij}(c_{vi}\,(1-c_{vj}))f_{uv} + \sum_{u=1}^{m}\sum_{i=1}^{n}\sum_{j=1|i<j}^{n} 2DSM_{ji}\,h_{ui} \quad (2)$$

$Subject\ to$

$$\sum_{v=1|v>u}^{m} x_{uv} + \sum_{i=1}^{n} y_{ui} = 2 \qquad \forall\, u = 1,\dots,m \quad (3)$$

$$\sum_{u=1}^{m} y_{ui} = 1 \qquad \forall\, i = 1,\dots,n \quad (4)$$

$$\sum_{u=1|v>u}^{m} x_{uv} = 1 \qquad \forall\, v = 2,\dots,m \quad (5)$$

$$c_{ui} - \sum_{v=1|v>u}^{m} c_{vi}x_{uv} + y_{ui} = 0 \qquad \forall\, u = 1,\dots,m;\ \forall\, i = 1,\dots,n \quad (6)$$

$$l_u - \sum_{i=1}^{n} c_{ui} = 0 \qquad \forall\, u = 1,\dots,m \quad (7)$$

$$l_u - l_{u+1} \geq 0 \qquad \forall\, u = 1,\dots,m-1 \quad (8)$$

$$q_u - \min(l_u - l_{u+1}, 1) = 0 \qquad \forall\, u = 1,\dots,m-1 \quad (9)$$

$$f_{uu+1} - q_u = 0 \qquad \forall\, u = 1,\dots,m-1 \quad (10)$$

$$\sum_{v=2}^{m} c_{vi}\,f_{uv} + h_{ui} - q_u = 0 \qquad \forall\, u = 1,\dots,m;\ \forall\, i = 1,\dots,n \quad (11)$$

$$q_v\left(\sum_{u=1}^{m} f_{uv+1} - 1\right) >= 0 \qquad \forall\, v = 1,\dots,m-1 \quad (12)$$

$$f_{uv}f_{uw}x_{kw}(l_k - l_v - 1) \geq 0 \qquad \forall\, u = 1,\dots,m;\ \forall\, v = 1,\dots,m;\ \forall\, k = 1,\dots,m;\ \forall\, w = 1,\dots,m|w \neq v \quad (13)$$

$$f_{uv}h_{ui}\,y_{wi}(l_w - l_v - 1) \geq 0 \qquad \forall\, u = 1,\dots,m;\ \forall\, v = 1,\dots,m;\ \forall i = 1,\dots,n;\ \forall\, w = 1,\dots,m|w \neq v \quad (14)$$

The objective function (Eq. 2) seeks the minimum value of the summation of Modularity Index (MI) over all levels of the product structure tree; i.e. the Overall Modularity Index (OMI). The DSMs used in this research are assumed to be symmetric; hence, the used expression calculates the OMI for one side of the DSM (above or under the matrix diagonal) and multiplies it by 2. However, the objective function can be easily adjusted to consider non-symmetric DSM instances.

Four sets of constraints are used to ensure the generation of valid product structure trees representing feasible solutions. Accordingly, the first set of constraints (Eq. 3) ensures that each node is the combination of exactly two branches; either two other nodes, two components or one other node and one component. The second set of constraints (Eq. 4) states that each component must branch out directly from one single node; no component could be branched out of two or more different nodes at the same time. Similarly, the third set of constraints (Eq. 5) ensures that every node, except the root node, is directly branched out from another node; no node could be branched out from two or more different nodes at the same time. The fourth set of constraints (Eq. 6) defines how the c array is extracted from the x and y arrays used in the first three sets of the constraints to ensure the structural feasibility of generated trees.

The sets of constraints given by Eq.'s 7 to 14 are used to define the clusters formed at each level of the product tree to be utilized by the objective function to calculate the MI at all levels of the tree. The set of constraints given by Eq. 7 determines the value of the variable l which defines the level of a given node represented by number of components it includes. The set of constraints given by Eq. 8 state that rows in the c array that belong to higher tree levels should appear first, in terms of their order in the array. This is to simplify comparison of the various levels of tree nodes.

The number of tree levels is not known a priori as mentioned earlier, however, it cannot exceed the number of the tree nodes (excluding the root node). Thus, it is assumed that a maximum of (m-1) tree levels could exist. The set of constraints given by Eq. 9 identifies which of those assumed levels will ultimately exist by comparing the difference in levels between every pair of consecutive nodes, since nodes are already ordered according to set of constraints given by Eq. 8.

The set of constraints given by Eq. 10 ensures that for every tree level an assignment of tree nodes into modules will take place. The set of constraints given by Equation 11 ensure that the modularization done at any tree level is considering all the components, i.e. to avoid partial or incomplete clustering of components at each tree level. The set of constraints given by Eq. 12 make sure that formed levels as well as the modules identified at those levels are different.. This is achieved by forcing one of the nodes at each level to be a module to its corresponding level only.

Nodes to be selected as modules at a given tree level cannot be sub-modules of other modules at that level. For instance, the node containing the two components 6 and 8 in the tree shown in Fig. 2 cannot be considered a module at level 3 of that tree. The same condition applies to individual components. The set of constraints given by Eq. 13 prevent such an invalid clustering from occurring by comparing levels of nodes forming clusters at each tree level; while the set of constraints given by Eq. 14 serve the same purpose for individual components.

4.3. Linearization of the objective function

The formulated model contains non-linear objective function and constraints. Only the objective function is linearized as it was found that linearizing the constraints would significantly increase the number of decision variables and reduce the capacity (i.e. number of product components) that can be handled by the model. The objective function is a cubic function that includes only binary variables. It can be simply linearized by defining a new binary decision variable to replace the multiplication of each pair of binary decision variables. Accordingly, two new sets of decision variables are defined as follows:

$cf1_{uvi}$ A binary element valued at 1 if the two binary variables f_{uv} and c_{vi} are to have the value 1

$cf2_{uvij}$ A binary element valued at 1 if the two binary variables $cf1_{uvi}$ and c_{vj} are to have the value 1

In addition, six auxiliary sets of constraints (not reported here for size limitations) are also added to the model to establish mathematical relationships between the new linearization decision variables and the original of decision variables. The new linearized objective is shown in function in Eq. 2'.

$$Minimize \text{ OMI} = \quad (2')$$

$$\sum_{u=1}^{m}\sum_{v=1}^{m}\sum_{i=1}^{n}\sum_{j=1|i<j}^{n} 2(1-DSM_{ij})cf2_{uvij} + \sum_{u=1}^{m}\sum_{v=1}^{m}\sum_{i=1}^{n}\sum_{j=1|i<j}^{n} 2DSM_{ij}(cf1_{uvi} - cf2_{uvij}) + \sum_{u=1}^{m}\sum_{i=1}^{n}\sum_{j=1|i<j}^{n} 2DSM_{ji}\,h_{ui}$$

5. Illustrative Example

The developed non-linearly constrained integer programming model is written in AMPL (A Mathematical Programming Language) and solved using the SCIP MIP (Mixed Integer Programming) software package [26] which is available through the NEOS server (www.neos-server.org/neos/solvers) hosted by the Wisconsin Institute for Discovery. The hand-held massager example (Fig. 1) is used to demonstrate the proposed overall modularity concept and the developed mathematical model. The massager casing is excluded from the analysis because it is the base component that contains all other components. The only input needed by the developed model is the DSM in Fig. 1 representing interactions between components of the massager.

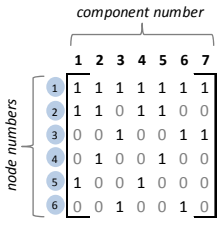


Fig. 5. Optimal *c* matrix obtained by the model

The optimal product tree represented in matrix form (the *c* array) for obtaining the optimal overall modularity (i.e. minimum OMI) is shown in Fig. 5. Decoding the *c* array into product tree form using PHYLIP 3.69 (www.phylip.com), which is a free access software for inferring and analyzing phylogenetic trees, is described in [25].

The decoded massager structure tree is shown is Fig. 6. This is the full modularity map for the entire massager components that identifies sub-modules inside each module. Accordingly, for the massager to have optimal OMI it should have three hierarchical levels. At the highest level of the tree, the massager is arranged into two modules: 1) the motor module (motor, vibrating head, bracket and lever) and the power inlet module (power cable, switch and wiring). At the second level the motor module is further decomposed into two sub-modules, the vibrating head with the supporting lever as a sub-module and the motor with the bracket as another sub-module. At the lowest level of the tree, the power inlet module is further decomposed into two sub-modules; the power cord with the on/off switch as sub-module and the wiring individual component as another module.

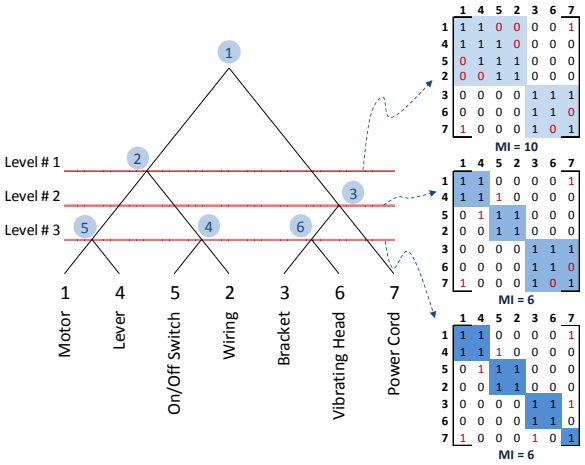


Fig. 6. Obtained optimal product structure tree showing the corresponding clustered DSM at each tree level

The resulting clustered DSM is constructed by re-ordering the components of the original DSM according to the order in which they appear on the tree. Three clustered DSMs (a DSM for each tree level) are shown on the left side of the obtained product tree in Fig. 6. Clustered DSMs at each tree level would not appear different from each other in terms of their composition (positions of zeros and ones inside the matrix), however, the difference is the way clusters of components are defined in each DSM which depends on the components clustering into nodes at different tree levels. The value of objective function (OMI) is 22 which is the sum of the MI for the 3 levels of the product tree; MI is 10 for the first level and 6 for level two and three.

The only output from the model that the user would be interested to look at is the *c* array shown in Fig. 5. It encodes the resulting optimal product structure tree which is the ultimate objective of the model. The resulting values for the *f* and *h* arrays, as well as other variables used by the model, are

all internal variables that do not concern the user, however, they are reported here for completeness. The resulting *q* vector defining which of the initially assumed tree levels is to actually exist is as follows: [1 1 1 0 0 0]. Only three elements were assigned the value "1" meaning that only 3 levels exist as shown in the resulting tree in Fig. 6. The resulting values for the *f* and *h* arrays defining the modules at each tree level are shown in Fig. 7. The *f* array shows the nodes forming modules at each tree level; while the *h* array shows individual components that represent modules (if any) at each level.

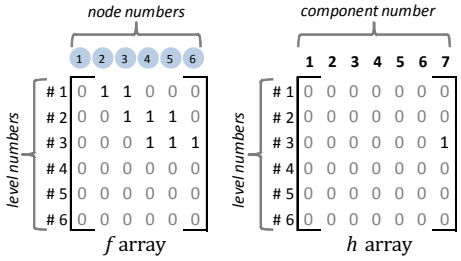


Fig. 7. Obtained values for the f and h arrays

6. Conclusion

This paper addressed the design of modular product architecture from a new perspective by aiming at optimizing the product architecture for optimal overall modularity. The notion of overall modularity is introduced, for the first time, and is implemented using a novel non-linearly constrained integer programming model. Given the DSM representing on/off physical interactions between various components of a certain product, the model builds the corresponding product structure tree with the optimal modularity over all its hierarchical levels - Overall Modularity Index (OMI) - without prior knowledge of the number of those levels or the number of modules at each level. Every potential module of a given product is considered as a product on its own, where its architecture needs to be designed for optimal modularity. The proposed modular product architecture design method is demonstrated using a hand-held body massager example.

Future research would consider not only spatial interaction but also multiple forms of interaction between components such as the existence of electric connection between two components that are not necessarily physically adjacent. Quantitative information such as the strength or importance of interaction between components would also be considered. Furthermore, methods to increase the ability of the presented mathematical programming model to handle large size problems (products with tens of components) should be explored.

References

[1] Ulrich, K., 1995,The role of product architecture in the manufacturing firm, Research policy, 24/3: 419-440.
[2] Yu, T.-L., Yassine, A.A., and Goldberg, D.E., 2007,An information theoretic method for developing modular architectures using genetic algorithms, Research in Engineering Design, 18/2: 91-109.
[3] Kremer, G.E.O. and Gupta, S., 2013,Analysis of modularity implementation methods from an assembly and variety viewpoints, The International Journal of Advanced Manufacturing Technology, 66/9-12: 1959-1976.
[4] Baldwin, C.Y. and Clark, K.B., 2000, Design rules: The power of modularity, Vol: 1, MIT Press.
[5] Shamsuzzoha, A., 2011,Modular product architecture for productivity enhancement, Business Process Management Journal, 17/1: 21-41.
[6] Baldwin, C.Y. and Clark, K.B., 2003,Managing in an age of modularity, Managing in the Modular Age: Architectures, Networks, and Organizations, 149.
[7] Gershenson, J., Prasad, G., and Zhang, Y., 2003,Product modularity: definitions and benefits, Journal of Engineering Design, 14/3: 295-313.
[8] Shamsuzzoha, A., Helo, P.T., and Kekale, T., 2010,Application of modularity in world automotive industries: a literature analysis, International Journal of Automotive Technology and Management, 10/4: 361-377.
[9] Gershenson, J.K., Prasad, G.J., and Zhang, Y., 2004,Product modularity: measures and design methods, Journal of Engineering Design, 15/1: 33-51.
[10] Allen, K.R. and Carlson-Skalak, S., 1998,Defining product architecture during conceptual design, A Proceedings of ASME, Atlanta, GA, Paper No. DETC98/DTM-5650.
[11] Ericsson, A. and Erixon, G., 1999, Controlling design variants: modular product platforms, Society of Manufacturing Engineers.
[12] Stone, R.B., Wood, K.L., and Crawford, R.H., 2000,A heuristic method for identifying modules for product architectures, Design studies, 21/1: 5-31.
[13] Zhang, W., Tor, S., and Britton, G., 2004, A functional modelling approach for modular product design. in International Conference on Manufacturing Automation: Advanced Design and Manufacturing in Global Competition.
[14] Huang, C.-C. and Kusiak, A., 1998,Modularity in design of products and systems, Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on, 28/1: 66-77.
[15] Ko, Y.-T., 2013,Optimizing product architecture for complex design, Concurrent Engineering, 21/2: 87-102.
[16] Steward, D.V., 1981,Design structure system: A method for managing the design of complex systems, IEEE TRANS. ENG. MGMT., 28/3: 71-74.
[17] Eppinger, S.D. and Browning, T.R., 2012, Design structure matrix methods and applications, MIT press.
[18] Tilstra, A.H., Seepersad, C.C., and Wood, K.L., 2012,A high-definition design structure matrix (HDDSM) for the quantitative assessment of product architecture, Journal of Engineering Design, 23/10-11: 767-789.
[19] Browning, T.R., 2001,Applying the design structure matrix to system decomposition and integration problems: a review and new directions, IEEE Transactions on Engineering Management, 48/3: 292-306.
[20] AlGeddawy, T. and ElMaraghy, H., 2013,Optimum granularity level of modular product design architecture, CIRP Annals-Manufacturing Technology, 62/1: 151-154.
[21] AlGeddawy, T. and ElMaraghy, H., 2013, Determining Granularity Level in Product Design Architecture, in Smart Product Engineering, Springer, 535-542.
[22] AlGeddawy, T. and ElMaraghy, H., 2015,Determining Granularity of Changeable Manufacturing Systems Using Changeable Design Structure Matrix and Cladistics, ASME Journal of Mechanical Design, 137/4: 041702-01 - 041702-12.
[23] Page, R.D., 2003, Tangled trees: Phylogeny, cospeciation, and coevolution, University of Chicago Press.
[24] Pandremenos, J. and Chryssolouris, G., 2011,A neural network approach for the development of modular product architectures, International Journal of Computer Integrated Manufacturing, 24/10: 879-887.
[25] Kashkoush, M. and ElMaraghy, H., 2015,Knowledge-based model for constructing master assembly sequence, Journal of Manufacturing Systems, 34: 43-52.
[26] Achterberg, T., 2009,SCIP: solving constraint integer programs, Mathematical Programming Computation, 1/1: 1-41.