A General Framework for Combined Module-and Scale-based Product Platform Design

Fabrizio Marinelli, Olivier de Weck, Daniel Krob, Leo Liberti

A General Framework for Combined Module- and Scale-Based Product Platform Design

Fabrizio Marinelli* **Olivier De Weck†**

Daniel Krob, Leo Liberti‡

Abstract

The modelling framework for product variety design proposed herein caters to both module-based and scale-based platforming and carries out commonality optimization on a set of product variants having different functional architectures. The product family architecture is modeled by a extended functional breakdown structure and valid design synthesis across the variants is achieved by instrumenting an attribute propagation mechanism from customer requirements through functional modules to physical components. The model is based on a set of variables and constraints which describe both component selection and attribute value commonality and which can be used in conjunction with various objective functions, depending on the market model employed.

1 Nomenclature

Sets

P	product family, i.e., a set of product variants
Q	set of physical components used to build the product variants
V	set of functional modules
W	set of functional modules and physical components, i.e.,

*Università Politecnica delle Marche, Dipartimento di Ingegneria Informatica, Gestionale e dell'Automazione, via Brecce Bianche, I-60131 Ancona, Italy. Email: marinelli@diiga.univpm.it

†Dept. of Aeronautics and Astronautics and Engineering Systems Division, MIT Cambridge, Massachusetts, 02139 Email: deweck@mit.edu

‡LIX, École Polytechnique, F-91128 Palaiseau, France Email: {krob, liberti}@lix.polytechnique.fr

	$W = V \cup Q$
F_s	set of continuous customer attributes
F_d	set of discrete customer attributes
F	set of customer attributes; $F = F_s \cup F_d$
$F(v)$	set of customer attributes relevant to module v, $v \in V$
A_s	set of continuous functional or physical attributes
A_d	set of discrete functional or physical attributes
A	set of functional or physical attributes; $A = A_s \cup A_d$
$A(u)$	set of functional or physical attributes relevant to module or component u, $u \in W$

Architecture tree

$G = (V, E)$	tree graph modelling the product family architecture
v_0	root node of G; $v_0 \in V$
$N(v)$	set of neighbors of v; for all $v \in V, N(v) = \{u \in V \mid (v, u) \in E\}$
L	set of the leaf nodes of G; $L = \{u \in V \mid N(u) = \emptyset\}$
$I(v)$	subset of physical components that can be used to implement the leaf functional module $v \in L$; $I(v) \subseteq Q$

Parameters

λ^i_{vh}	requirements for the h-th customer attribute of module v in product variant i, $i \in P, v \in V, h \in F(v)$ (λ^i_{vh} is a subset of a given domain Λ^i_{vh})
ϑ^q_k	design bounds for the k-th physical attribute of the q-th physical component, $q \in Q, k \in A(q)$
M	large enough constant (used for modelling disjunctive constraints)

Variables

y^i_{uk}	continuous: value of the attribute k of module or physical component u in product variant i
w^{ij}_{qk}	binary: 1 if product variants i and j use same component q with same value for attribute k
x^i_v	binary: 1 if module v is used in product variant i
p_q	binary: 1 if q is a common component
z^i_{vq}	binary: 1 if physical component q implements elementary module v in product variant i

r_q^i binary: 1 if component q is used at least once in product i

Functions

ψ_{vh}^i requirement functions:
$\psi_{vh}^i : \mathbb{R}^{|A|} \to \Lambda_{vh}^i$, for all $i \in P, v \in V, h \in F(v)$

δ_{vk} composition functions:
$\delta_{vk} : \mathbb{R}^{|N(v)|} \to \mathbb{R}$, for all $v \in V \setminus L, k \in A(v)$

φ_v module interface functions:
$\varphi_v : \mathbb{R}^{|A|(|V|-1)} \to \{0, 1\}$, for all $v \in V$

χ_q component interface functions:
$\chi_q : \mathbb{R}^{|A|(|L|-1)} \to \{0, 1\}$, for all $q \in Q$

We emphasize notation by always using i, j for product variants in P, u, v for functional modules in V, h for customer attributes in F, k for functional/physical attributes in A and q for physical components in Q.

2 Introduction

The top-down platform approach (*proactive platforming*, see [24]) for the design and development of a new product family can be broadly decomposed into three stages, see [13]:

- *Product definition* which deals with the overall product positioning strategy. The behavior of individual and groups of customers is investigated from the marketing perspective, i.e., in terms of needs, preferences and degree of satisfaction. The product family is then described by a market segmentation grid which embodies a product variant for each market niche, and the customer requirements of each product variant are mapped, at various level of abstraction, into the functional features that the variant should have. The product definition seeks for different functional variety implementations which emphasize the uniqueness of product variants and best fit with the market segmentation grid.

- *Product design* which basically consists in a mapping process from the hierarchy of functional features to the hierarchy of design parameters and physical components. The mapping describes how the functional features are implemented and takes into account both engineering concerns and available product technologies.

- *process and supply chain design* which concerns costs of manufacturing, assembly structure and resource allocation in the logistic domain.

In order to tradeoff between development costs and product performance deviation from individual optima, product design exploits, at different conceptual levels of the design process, both functional feature commonality and physical component commonality [20].

At the architectural level of the design process, the main concern is to identify the shared features and the cohesive functional architecture underlying the product family. Indeed, each variant could have the own functional structure since either (*i*) a customer requirement of a given market niche could be translated to a functional feature which is not required in the other variants of the product family or (*ii*) different product quality degrees required from different customers could correspond to the same functional feature but with different performance attributes which in turns presuppose different technologies and hence different functional subsystems.

Once the functional architecture of the product family has been established, each variant may be instantiated by quantifying design parameters within a given physical component configuration (*scale-based* platforming), by combining existing components each one completely specified in term of design parameters (*module-based* platforming), or both. In the former case, commonality is achieved by defining a set of common design parameters across the variants of the product family, whereas in the latter case the platform consists in a set of common physical components.

Usually, optimization methods are only adopted in the instantiation level of the design process where the functional architecture of the product family is considered as given. Moreover, except few cases, module-based and scale-based platforming are carried out separately. However, both the above decompositions, i.e., functional vs. physical commonality and module attributes vs. module combinations optimization, suffer of several weakness.

About the former, although computational optimization is in general easier when the product family architecture is given beforehand, the choices on the functional structure of the variants deeply affect the product variety optimality. Moreover, the functional architecture of a variant could even be unknown in advance since in general the customer tradeoffs among product features, i.e., among product functional representation. Therefore, a functional commonality problem arises beside the well-studied component commonality problem.

For what concern scale- and module-based platforming decomposition, it leads, as already observed by Fujita [7], to a egg-and-chicken situation.

Indeed, except of very few cases where module- and scale-based platforming can be applied separately, e.g., well-standardized and technology-driven products, design parameters quantification supposes a well-defined set of physical components, and conversely the selection of a combination of physical components is clearly affected by the content of components. The simultaneous optimization of both design parameters and component combinations instead has the advantage of achieving economy of scale of components and increase the performance of the product family, since the commonalization of design parameters makes physical components more similar to each other and aim to standardize hidden subsystems or parts behind components and therefore to reduce costs in the manufacturing domain.

In this paper, we propose a modelling framework for product variety design that integrates module-base and scale-based platforming and carries out commonality optimization on a set of product variants having different functional architectures. The framework consists of decision variables and constraints expressed as a mathematical programming formulation. This model carries out commonality optimization with respect to both continuous and discrete component attributes and product functionalities. The output of the model consists in a selection of common components that should be part of the platform and an assignment of values to the attributes of selected components. We also show how several meaningful objective functions and platform evaluation metrics addressed in the literature can be expressed by means of the model variables.

Each variant of the product family is described by two interlinked hierarchical structures, the *functional breakdown structure* (FBS), and the *physical breakdown structure* (PBS), and the product family architecture is described by an extended FBS (e-FBS) as in Fig. 2. In our modelling framework, we are interested in matching the elementary functions to components and in determining attribute values, see Fig. 4.1. We distinguish several categories of product attributes. *Physical components* are non-decomposable parts with a separate identity as typically indicated by a part number [6]. These are characterized by their *physical attributes* such as e.g. size, color, etc.

For example, the FBS of an electrical tootbrush can be described by the root functional module "teeth cleaning", further decomposable into "manipulating", "operating", etc. The PBS's root node would be the toothbrush itself, composed of the housing, motor, etc. The *product family architecture* is given by the merging of the constituent architectures of the individual variants.

Module characteristics, such as the price, are described by *functional attributes*. *Customer attributes*, such as the endurance, impose requirements

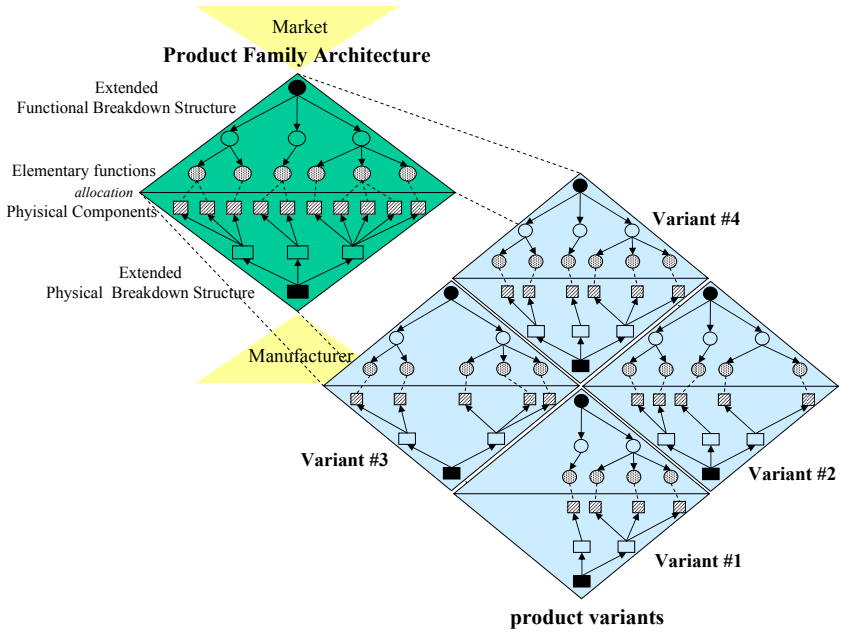

Figure 1: Functional and physical breakdown structure of product family architecture and product variants.

on the functional attributes by market segment. Attributes can either belong to one or more amongst physical, functional or customer domain (e.g. the weight can be a customer, functional or physical attribute). In this case, the same attribute name will be used in different contexts. The key feature of the proposed modeling framework is the propagation of different attributes from customer through function to physical. This is implemented by means of dependency relations between the different types of attributes and levels of decomposition. For example, the values of the physical attributes "weight" of each physical component associated to a certain functional module are added up to define the functional attribute "weight" of the module; this value must then satisfy the requirements imposed on the corresponding customer attribute "weight". Such relations can be linear or nonlinear analytic expressions as well as black box functions. Weight provides an example of a linear relation, Euclidean dimensions that of a nonlinear relation; the relation between the physical attribute "motor_power", the functional attribute "brushing rate" and the customer attribute "price" is possibly best modeled

as a black box.

In this paper we assume we know (i) the functional and physical product family architecture, together with the sets of physical, functional and customer attributes, (ii) the number of product variants belonging to the product family, and (iii) the *customer requirements* i.e. the ranges of customer attributes for each market segment. We remark that we are mainly concerned with product platform formation rather than tactical integrated product/process platform problems, however the proposed framework ensures physical component compatibility even if we do not investigate the manufacturing issues in details.

The rest of this paper is organized as follows. In Section 3 a product platform design framework which unifies module- and scale-based platforming and is based on a e-FBS representation of the product family is presented and discussed amongst related works. Section 4 describes in detail the features of such framework. Section 5 describes the relevant mathematical formulation in terms of decision variables, constraints and objective functions. Conclusions are drawn in Section 7.

3 Contribution and related works

In general a product architecture describes: (i) the hierarchical arrangement of functionalities and components into modules and submodules, (ii) the allocation of physical components to elementary functions, (iii) the functional interfaces among modules and (iv) the physical interfaces among components. Depending on the degree of coupling of functional and physical interfaces systems range from having a totally *modular* product architecture (independent attributes) to a totally *integral* product architecture (complex and coupled attribute interfaces), see [26].

Although virtually no product architecture is entirely modular or integral, most of the references in the literature only deal with either module combination optimization (module-based platforming) or module attribute optimization (scale-based platforming). In the former approach, e.g. see [21, 22, 25], the values of component attributes are fixed before optimization and product variants are instantiated by adding and/or removing one or more components from a set of common ones, i.e., from the platform. In the latter approach, e.g. see [4, 15, 18], commonality is achieved by scaling one or more component attributes; the platform is the set of attributes having common values across all the variants of the family.

Module-based platforming promotes supplementary value-adds compe-

tition as well as scale-based platforming is suitable for technology-driven products but in general an integrated approach is more desirable. Despite of this, as reported by Simpson [23], only few module- and scale-based platforming integrated approaches, see Hernandez et al. [12, 11] and Fujita and Yoshida [9], have been proposed as computational design optimization methods.

Hernandez et al. describe a general top-down platform design approach where each variant of the product family is regarded as a point in the space of customization, i.e., in the geometric space defined by the feasible combinations of all the feasible functional feature values. They formulate the problem of designing customizable products as an access problem in the space of customization and hierarchically solve it by a multi-stage optimization approach.

Fujita and Yoshida propose a unified framework based on the identification of three different types of components: *common*, *similar* and *unique*. In a common component, all the physical attributes take common values across all members of the product family. In similar components there is a subset of physical attributes taking common values across a subset of variants; similarity could be necessary when a physical attribute can not spread the large spectrum of values of a relevant customer requirement. In a unique component, each physical attribute always takes distinct values for each product variant of the family. Observe that similarity among components describes, in a different perspective, the multi-platforming concept addressed by other authors, see [12, 3, 14, 2]. Indeed, the number of platforms in a multi-platforming approach indicates the number of distinct values that a common design variable takes across the variants of the product family, each value defining a subset of similar components.

However, both the above references do not address the functional architecture variety and carry out some kind of decomposition in the optimization phase, sequential the former, hierarchical the latter. In particular, in [9] the functional architecture is assumed as given and each product is composed of a series of module slots where modules are installed, and for each slot a set of variants are available. Scalar-based platforming is implemented in a two-stage procedure: first, a suitable set of common scalar design variables, i.e., the platform, is chosen by a genetic algorithm and then the values are obtained via branch-and-bound and quadratic programming.

On the basis of Fujita et al. framework, both attribute and component commonality/similarity is achieved in our model by considering equality (or ε-difference for continuous attributes) of component types and component attributes. Satisfaction of customer requirements is obtained by mapping

physical attribute values of components to functional attribute values of modules through the product architecture tree up to the root node which represents the final product. At each tree node, we enforce the node functional attributes on the propagated functional attribute values and we require the satisfaction of customer requirements. When the mapping between functional features and design parameters can be easily addressed through the product family architecture, our integrated model can replace the multistage approach proposed by Hernandez. Otherwise, it can be used as an evolved mode for managing product variety within the space of customization proposed by Hernandez. Summarizing, our approach mainly contributes to:

- express in sufficiently general mathematical terms the product family design problem, a task which usually prelude the employment of computational optimization techniques;
- make the cross-level attribute propagation problem independent by means of abstract functions whose implementation depends on the particular problem structure;
- provide a unified view of physical and functional architecture as well as of module-bases and scale-based commonality;
- extend commonalization from physical components perspective to functional features domain.

4 The modelling framework

In this section we discuss the proposed modeling framework in detail. The concepts and mathematical notation are supported by a simple example concerning the battery powered toothbrush illustrated in Fig. 4. A physical and functional decomposition of the product is represented in Fig. 4 by means of an object-process diagram, see [1].

4.1 Assumptions

The model rests on the following assumptions.

- The number of variants in the family is fixed: each product corresponds to a market niche which is part of a given market segmentation grid [19].

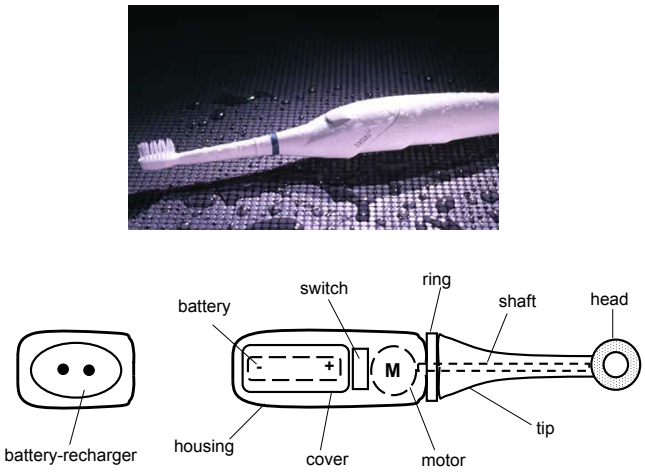

Figure 2: The battery powered toothbrush product.

- A given set of customer attributes identifies those product characteristics that cause different degrees of satisfaction among customers and influence their choice, and a range of feasible values (the customer requirements) is given for each customer attribute of each product in the family. We consider customer attributes in a broad sense: other than merely physical properties of a product, such as the weight or the color, a customer attribute could also be a functional characteristic, e.g. the ability to uniquely identify a toothbrush.

- A given set of functional/physical attributes describes the functional module/physical component characteristics, and a range of feasible values, called *design bounds*, is given for each attribute of each physical component which can be allocated to some product variant of the family.

- The product family is described by means of a *product family architecture* which is an *extended* FBS/PBS (e-FBS/PBS). Each product variant is thought of as a subgraph of the product family architecture describing both the set of functional modules that the product must provide (and therefore the set of functionalities), and the set of physical components allocated to such functional modules (see Fig. 2

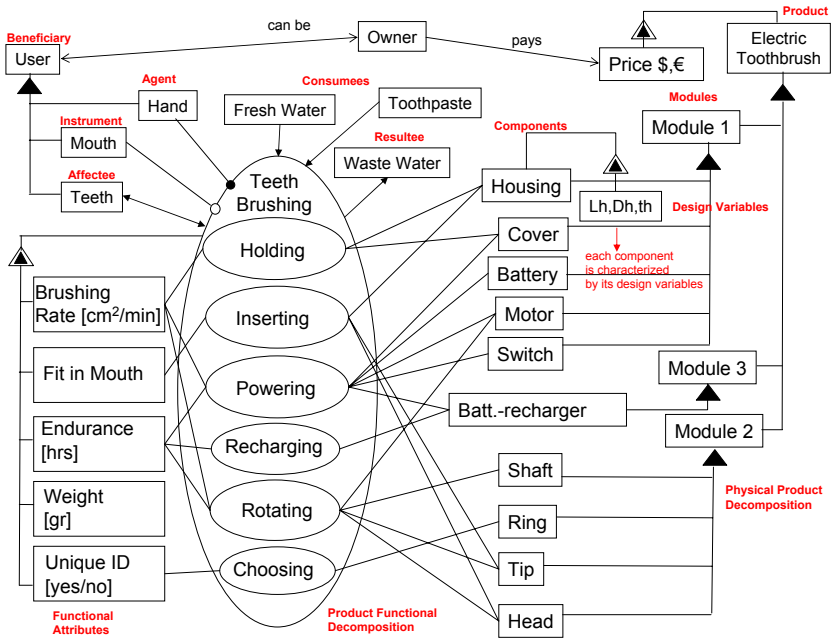

Figure 3: Toothbrush object-process diagram.

and Fig. 4.1). In the following we only focus on the e-FBS part of the product family architecture; details of manufacturing are addressed by means of component interface functions, see Section 4.5. Commonality is achieved either by using common components (see components A and C in Fig. 4.1) or by sharing attribute values (see attribute 3 of component E in Fig. 4.1).

4.2 Sets

Let P be the set of variants in a product family and Q the set of physical components used to assemble the product variants in P.

Customer preferences are described by a set F of customer attributes, whereas module and component characteristics are described by a set A of functional/physical attributes. Customer and functional/physical attributes can be either continuous values, e.g. the weight or the endurance, or discrete values, e.g. the color or the battery type. Therefore we assume that $F =$

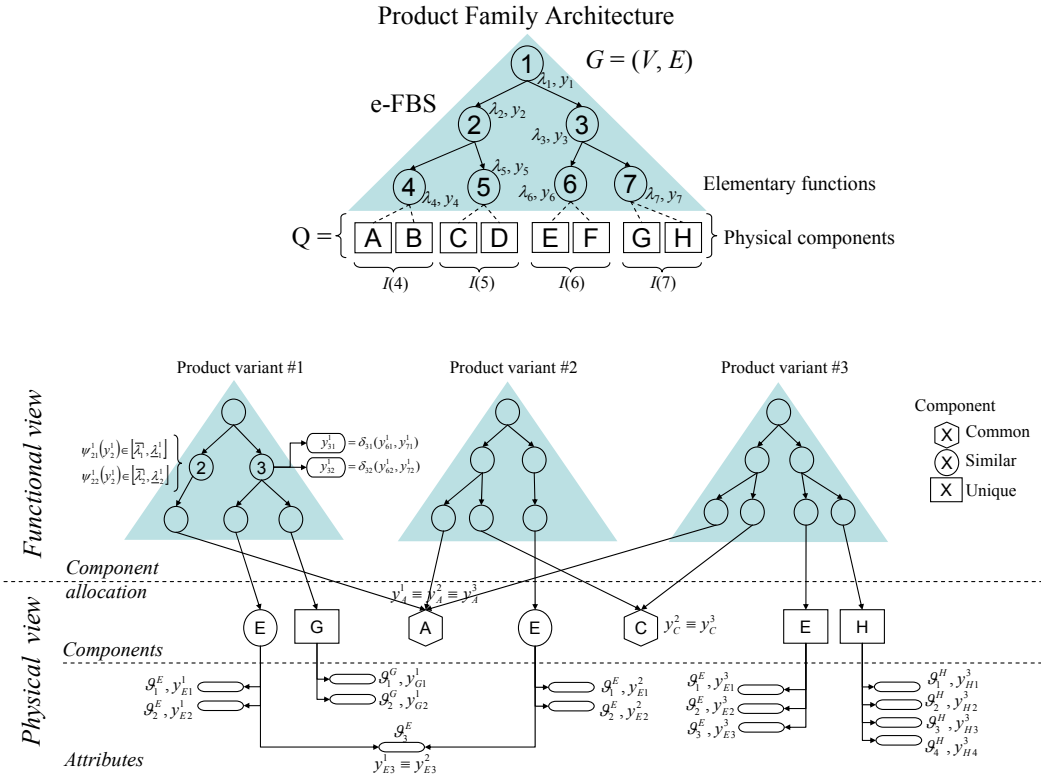

Figure 4: product family architecture.

$F_s \cup F_d$ and $A = A_s \cup A_d$ where F_s (A_s) and F_d (A_d) are respectively the set of scalar and discrete customer (functional/physical) attributes.

Example 4.1 *The market segmentation grid of the toothbrush of Fig. 4 is reported in Tab. 4.1. The product family P consists of 6 variants* $\{L_1, L_2, L_3, N_1, N_2, N_3\}$. *Fig. 4 also shows the physical parts constituting the toothbrush. They fall in two categories: those manufactured by changing the design parameters of a base component (scale-based design), and those selected among different models (model-based design). Shaft and cover, for example, belong to the former category since different shafts can be obtained by varying the length of the mould. The head, instead, is chosen between the linear and the circular model and therefore falls in the latter type of part. Clearly, a more complex situation in which both model selection and parameter setting are performed*

Table 1: TOOTHBRUSH MARKET SEGMENTATION GRID.

	women and children (small mouth)	men (large mouth)
Frequent brushers (2-3 times daily)	N_1	L_1
Normal brushers (once daily)	N_2	L_2
Travelers	N_3	L_3

can be considered. In our example, the set of available physical components are $Q = \{$*ring, housing, cover, switch,* $battery_A$*,* $battery_B$*, motor, tip, shaft,* $head_A$*,* $head_B$*, recharger*$\}$*.*
From the customer perspective, the toothbrush is mainly described by the set of attributes $F = \{$*price, weight, endurance, efficacy, uniqueness, fitness*$\}$*.* $Uniqueness$*, i.e., the property of being recognized among other toothbrushes, is a boolean attribute depending on the color of the ring;* $fitness$*, which depends on the size and the shape of the tip and the head, is a discrete attribute taking values in the set* $\{$*average, comfortable, fitting*$\}$*. Therefore* $F_s = \{$*price, weight, endurance, efficacy, uniqueness*$\}$*, and* $F_d = \{$*fitness*$\}$*. Notice that boolean attributes are treated as continuous one, see Section 4.4. The main functional/physical attributes are* $A_s = \{$*length, weight, motor_power, price, rechargeable*$\}$*, and* $A_d = \{$*battery_type, color, head_type*$\}$*. We remark that discrete attributes such as color could be instantiated by either a single element, e.g., the color of ring, or by a subset of the relevant domain, e.g., the different colors of housing, ring, tip, etc.*

4.3 Product family architecture

Functional and structural views for a single product typically consist in hierarchical data structures represented by rooted trees where the root is the finished product, the nodes are functional subsystems or assembled components and the leaves are functional or physical elementary components. The most common type for a manufacturing firm is the Bill-of-Material (BoM). Several solutions have been proposed in the literature to extend such representation to describe product family architectures. The best known are (*i*) the Generic Bill-of-Material (GBoM), introduced by Hegge and Wortmann [10], which allows all variants of a product family to be specified only once, and (*ii*) graph rewriting systems [5], based on graph grammar techniques and used for formal representation of product families and automatic generation of variants. However, both GBoM and graph rewriting systems,

although being valid descriptive and prescriptive tools in the domain of product modelling, suffer of inherent limitations when employed in an optimization framework since they are mainly conceived as representation tools rather than decision making optimization tools.

Let $G = (V, E)$ be the FBS part of a product family architecture where V is the union set of all the functional modules of the product family, and E is the set of arcs representing the functional breakdown relationship. G describes the arrangement of product functionalities into functional modules; in general it is a direct acyclic graph, see the example below, but it often takes the shape of a tree. We indicate with $v_0 \in V$ the root node of G, with $N(v)$ the set of neighbors of v, for each $v \in V$, and with L the set of the leaf nodes of G. Node v_0 corresponds to the functional module implementing all the required functionalities, i.e., to the end-customer product variants in P. The set $N(v)$ consists of the functional submodules obtained by decomposing module v. Notice that in a product family architecture not all the submodules of v must (or can) be part of v since the implementation of variants and options. Finally, L is the set of the *elementary* functional modules, i.e., those modules actually implemented by physical components in Q. In the following, for each $v \in L$, $I(v) \subseteq Q$ indicates the subset of physical components that can be used to implement the elementary functional module v. Notice that in general, the sets $I(v)$ define a cover of Q, i.e., a component can be allocated to more than one elementary module of each product.

Example 4.2 *(continued)*

In the following we map index sets P, F and A to sets of natural numbers of the same cardinality in order to keep index notation clear. Elements of Q and V are referred to by letters and numbers as reported in Fig. 4.2.

Figure 4.2 describes the FBS part of the toothbrush product family architecture. Teeth cleaning, represented by node v_0, is the overall functionality of the toothbrush. It can be decomposed into manipulating, operating and recharging functions, so $N(v_0) = \{v_1, v_2, v_3\}$. Module Recharging is optional, i.e., it can be included or not in the FBS of variants. Nodes v_4 and $v_9, \ldots, v_{17}$ correspond to elementary functions. Their interrelated arrangement highlights the degree of coupling of functional interfaces in this example. The component allocation to Battery and Head modules consists in selecting a suitable component in a set of available alternatives, i.e., $I(v_{12}) = \{E, F\}$ and $I(v_{16}) = \{K, L\}$. All other elementary modules are implemented by a single physical component, i.e., component allocation is performed by attribute scaling.

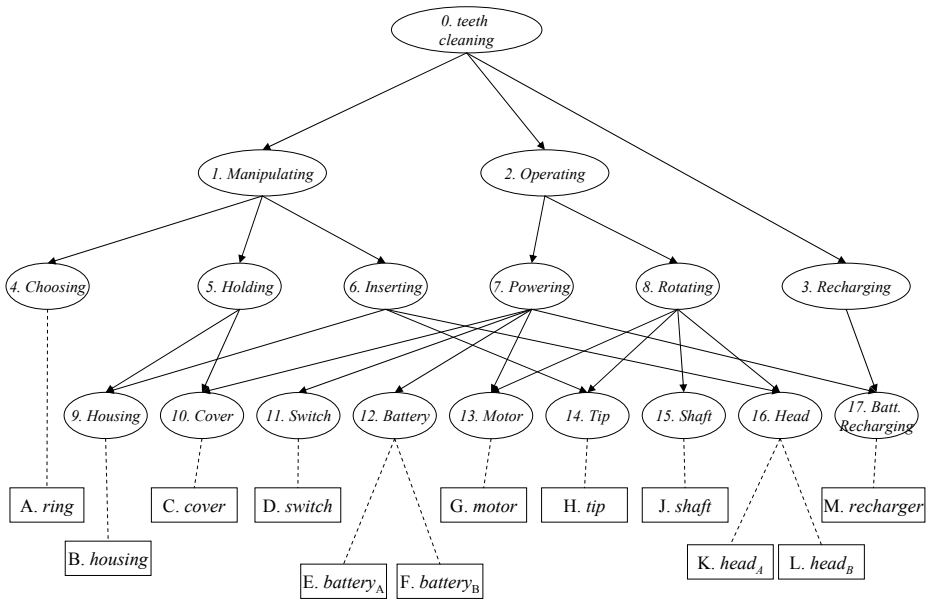

Figure 5: e-FBS of the toothbrush.

4.4 Numerical parameters

For each product variant i and functional module v of the architecture tree, let λ^i_{vh} be the requirements of the h-th customer attribute, i.e., the feasible values that the h-th customer attribute can take. The set λ^i_{vh} is an interval $[\underline{\lambda}^i_{vh}, \bar{\lambda}^i_{vh}] \in \mathbb{R}$ for scalar attributes and a discrete feasible set for discrete attributes, and is a subset of a given domain Λ^i_{vh}.

Moreover, for each physical component q, let ϑ^q_k be the design bounds for the k-th physical attribute, i.e., the feasible values that the k-th physical attribute of q can take. As above, the set ϑ^q_k is an interval $[\underline{\vartheta}^q_k, \bar{\vartheta}^q_k] \in \mathbb{R}$ for scalar attributes and a discrete feasible set otherwise.

In this paper we consider scalar and discrete customer and functional/physical attributes. Notice that a boolean attribute h can be easily expressed as a scalar one: $h = 1$ (respectively $h = 0$) means that the customer attribute is (respectively, not) required, whereas $0 \leq h \leq 1$ indicates that the attribute

h is optional. Similar considerations can be made for functional/physical attributes.

We assume without loss of generality that the discrete sets ϑ_k^q and λ_{vh}^i are sets of positive integers, and that discrete attributes always take scalar values. Indeed, the value taken by a discrete attribute h could in general describe a subset of the requirements (or the design bounds) for h. This subset, however, can in general be modeled as a binary vector $\mathbf{d}_h$ whose components set to one correspond to the requirements (or the setting) for h. The inclusion relationships between attribute sets can then be easily formulated as integer linear constraints on the above binary vectors.

Example 4.3 *(continued)*
Some customer requirements are:

- *children who frequently brush teeth (market niche N_2) prefer an effective and identifiable toothbrush: $\lambda_{04}^5 \geq 60\%$, and $\lambda_{05}^5 = 1$;*
- *male travelers (market niche L_3) opt for a light and durable toothbrush without taking care of uniqueness: $\lambda_{02}^3 \leq 50$ gr, $\lambda_{03}^3 \geq 50$ hrs, and $\lambda_{05}^3 \in [0, 1]$.*

Some design bounds are:

- *the length (and the weight) of the shaft must range in a given interval due to manufacturing technological constraints: $\vartheta_1^J \in [80, 120]$ mm, and $\vartheta_2^J \in [14, 20]$ gr;*
- *rings are available in three colours, i.e., $\vartheta_7^A \in \{red, blue, white\}$, whereas all tips and housings are white, i.e., $\vartheta_7^B = \vartheta_7^H = \{white\}$;*
- *batteries are available in two models: common household carbon-zinc or alkaline ($battery_A$) and li-polymer ($battery_B$). The former is cheaper but non rechargeable, i.e., $\vartheta_4^E \in [0.4, 0.7]$ \$, $\vartheta_5^E = 0$, and $\vartheta_6^E \in \{alkaline, carbon\_zinc\}$, whereas the latter is more expensive but rechargeable, i.e., $\vartheta_4^F \in [20, 40]$ \$, $\vartheta_5^F = 1$, and $\vartheta_6^F = \{polymer\}$.*

Notice that some attributes, such as rechargeable and motor_power, are relevant only for a subset of modules and/or components.

4.5 Abstract Functions

The product family architecture graph only partially provides the information needed to describe how the variants of the product family can be

correctly assembled. Usually in design, customer requirements, functional attributes and design bounds fall in distinct domains so that they are defined independently and are not necessarily compatible with each other. Hence, a full family description also requires some knowledge (*i*) on *requirement translation*, i.e., on how functional characteristics of a product result in customer preferences, (*ii*) on *module composition*, i.e., on how submodule attributes settle module attributes, and (*iii*) on *module and component interfaces*, i.e., on what are the compatibilities among the values taken by attributes, between functional modules and between physical components. all the above features are implemented in our framework by sets of abstract functions. In particular:

- for all $i \in P, v \in V, h \in F(v)$, the *requirement function* $\psi^i_{vh} : \mathbb{R}^{|A|} \to \Lambda^i_{vh}$ maps the values of the functional module attributes of v into the value of the h-th customer attribute. Notice that the requirement functions can be defined not only for v_0 but also for submodules, since customer requirements in fact can relate on characteristics of submodules.

- for all $v \in V \setminus L, k \in A(v)$, the *composition function* $\delta_{vk} : \mathbb{R}^{|N(v)|} \to \mathbb{R}$ maps the attributes of the submodules of v to the attributes of v; naturally, δ will be defined in such a way that the argument corresponding to $u \in N(v)$ will be ignored if $k \notin A(u)$, i.e. an attribute which is irrelevant on a submodule will not influence the value of δ.

- for all $q \in Q$, the *component interface function* $\chi_q : \mathbb{R}^{|A|(|L|-1)} \to \{0, 1\}$ decides whether the current assignment of attribute values of the selected physical component q is compatible with the attribute values of other selected components.

- for all $v \in V$, the *module interface function* $\varphi_v : \mathbb{R}^{|A|(|V|-1)} \to \{0, 1\}$ decides whether the current assignment of attribute values of module v is compatible with the attribute values of other modules.

The module interface functions, and similarly the component interface functions, are used to model compatibility and restrictions relating to subsets of modules. They are conceived as boolean functions: their arguments are the attribute values of the selected modules and they return 1 if both the current selection of modules and the assignment of values to attributes are compatible, and 0 otherwise. A special case of such functions are the AND/OR conditions employed in the Generic Bill-of-Material (GBoM) [10]

since they do not depend on the values taken by the module attributes. Indeed, an AND condition simply describes a module composition, i.e., that all the submodules of v must be used to implement v whatever are the values of their attributes, whereas an OR condition simply describes options through which one can derive product variants, i.e., exactly one of the submodules of v can be used to implement v.

The above definitions are very general; as such, they do not explicitly describe the actual form of the $\psi, \delta, \varphi, \chi$ functions. The time and space complexity for finding a feasible solution for the model mainly depends on the form of such functions. For example, composition functions may range from easily linearizable (e.g. a summation of attributes) to functions that cannot be expressed in closed algebraic form (e.g. δ_{vk} could be the result of an auxiliary optimization or simulation problem).

Example 4.4 *(continued)*

Since Price and Weight attributes are both customer and functional, simple requirement functions working on homogenous dimensions are sufficient to model the translation between customer and functional domains. On the other hand, non trivial requirement functions must be defined for Endurance, efficacy, uniqueness and fitness customer attributes. The endurance of a variant, for example, can be computed by ψ^i_{03} in terms of employed battery type. Analogously, requirement function ψ^i_{04} can return the efficacy of a variant in terms of head type and brushing rate (the latter depending on the motor_power functional attribute), and ψ^i_{05} can link the uniqueness to the color of the ring. Finally the fitness can be computed by ψ^i_{06} in terms of length of the shaft and type of the head. Observe that, function ψ^i_{06} depends from the market niche since fitness is a subjective customer attribute.

The overall structure of the toothbrush must be consistent. Since the extended FBS describes the product family, i.e., each variant in general consists of a subset of modules, module interface functions implementing AND/OR conditions must be defined. For example, function φ_0 enforces module v_0 to be composed by both modules v_1 and v_2, and, optionally, by module v_3. Additional interface constraints can occur. For example, the condition that a recharger is required if and only if a rechargeable battery is adopted can be implemented by function φ_7.

The values of functional attributes must be set throughout the extended FBS according to the allocation of components. In our example, the overall weight and price of the toothbrush are given by functions δ_{02} and δ_{04} respectively, that simply add up weights and prices of submodules. Composition function δ_{01} associated to length attribute is a little bit different since the

overall toothbrush length depends on only housing, tip and head component lengths. Finally, δ_{07} determines the color of the toothbrush as the union of ring, housing and tip colors.

5 A mathematical formulation

The model framework described in Section 4 can be formalized in terms of mathematical modelling by means of the following sets of variables, constraints and objective functions.

5.1 Decision variables

There are three kinds of variables in the model: the real variables $\mathbf{y}$ that describe the values taken by functional/physical attributes, the binary variables $\mathbf{x}, \mathbf{z}$ and $\mathbf{r}$ that model the selection of functional modules and physical components, and finally the binary variables $\mathbf{p}$ and $\mathbf{w}$ that count component and attribute commonality. More formally:

- For all $i \in P, u \in W, k \in A(u)$, let y^i_{uk} be the value associated with the attribute k of module or physical component u in product variant i.
- For all $i \in P, v \in V$, let $x^i_v = 1$ if module v is used in product variant i, and let $x^i_v = 0$ otherwise;
- For all $i \in P, v \in L, q \in Q$, let $z^i_{vq} = 1$ if component q implements elementary module v in product variant i, and let $z^i_{vq} = 0$ otherwise;
- For all $i \in P, q \in Q$, let $r^i_q = 1$ if component q is used at least once in product i, and let $r^i_q = 0$ otherwise;
- For all $q \in Q$, let $p_q = 1$ if component q is common, and let $p_q = 0$ otherwise;
- For all $i < j \in P, q \in Q, k \in A(q)$, let $w^{ij}_{qk} = 1$ if i, j use same component q with same value for attribute k, and let $w^{ij}_{qk} = 0$ otherwise.

We can consider ordered subsets of variables by contracting the relevant indices, e.g. $\mathbf{y}^i_v = (y^i_{v1}, \ldots, y^i_{v|A(v)|})$ for all $i \in P, v \in V$ and so on.

5.2 Constraints

The model constraints mainly relate to (i) module composition (ii) interface implementation and (iii) evaluation of commonality.

5.2.1 Module composition

For each variant i of the product family, a set of physical components must be selected and set up in such a way that all the customer requirements of i are satisfied and all the design bounds of the chosen components are fulfilled. To this aim, we introduce a set of constraints implementing a propagation device that translates, through the levels of the product architecture, component design bounds into customer requirements. In particular, for each product variant i the propagation device must guarantee that:

- the attribute values of each functional module are feasible with respect to the requirements of the corresponding customer attributes;
- the attributes of a module u are consistently obtained from the attributes of the modules constituting u;
- the attribute values of a selected elementary module are feasible with respect to the design bounds of the component which has been selected to implement it.

The first step is implemented by constraints (1) and (2): for each customer attribute h, the scalar (discrete) value yielded by the requirement function ψ^i_{vh} must belong to the interval (the set) of the relevant requirement.

$$\forall i \in P, v \in V, h \in F_s(v) \qquad \underline{\lambda}^i_{vh} \leq \psi^i_{vh}(\mathbf{y}^i_v) \leq \bar{\lambda}^i_{vh} \tag{1}$$

$$\forall i \in P, v \in V, h \in F_d(v) \qquad \psi^i_{vh}(\mathbf{y}^i_v) \in \lambda^i_{vh}. \tag{2}$$

The second step can be modeled by resorting to the composition functions δ_{vk}. The attribute values of a module depend on the attribute values of its selected submodules, see constraints (3), whereas the attribute values of an elementary module must correspond to those of the physical component used to implement it, see constraints (4).

$$\forall i \in P, v \in V \setminus L, k \in A \qquad y^i_{vk} = \delta_{vk}(\mathbf{y}^i_k \odot \mathbf{x}^i) \tag{3}$$

$$\forall i \in P, v \in L, k \in A(v) \qquad y^i_{vk} = \sum_{q \in I(v)} z^i_{vq} y^i_{qk} \tag{4}$$

Notice $\mathbf{y}^i_k \odot \mathbf{x}^i = (y^i_{ku_1} x^i_{u_1}, \ldots, y^i_{ku_\alpha} x^i_{u_\alpha})$ where $\alpha = |N(v)|$, and that each argument $y^i_{ku} x^i_u$ of δ_{vk} takes the value y^i_{ku} if submodule u is currently selected, and 0 otherwise.

The third step is implemented by constraints (5) and (6) which guarantee that the scalar and discrete attribute values of a chosen physical component

belong to the relevant design bounds.

$$\forall i \in P, q \in Q, k \in A_s(v) \qquad \underline{\vartheta}_k^q \leq y_{qk}^i \leq \bar{\vartheta}_k^q \tag{5}$$

$$\forall i \in P, q \in Q, k \in A_d(v) \qquad y_{qk}^i \in \vartheta_k^q. \tag{6}$$

Finally, each selected elementary module must be implemented by exactly one physical component:

$$\forall i \in P, v \in L \qquad x_v^i = \sum_{q \in I(v)} z_{vq}^i. \tag{7}$$

5.2.2 Interface implementation

The functional and physical interfaces mainly concern the compatibility between selected modules and components. Due the definition of interface functions (see §4.5), such compatibility can be simply modeled by logical implications: the functional module v (physical component q) cannot be selected, i.e., $x_v^i = 0$ ($z_{vq}^i = 0$), if the current module (component) selection and functional (physical) attribute setting are not compatible, i.e., $\varphi_v(\mathbf{y}^i\mathbf{x}^i) = 0$ ($\chi_q(\mathbf{y}^i\mathbf{x}^i) = 0$).

$$\forall i \in P, v \in V \qquad x_v^i \leq \varphi_v(\mathbf{y}^i \odot \mathbf{x}^i) \tag{8}$$

$$\forall i \in P, v \in L, q \in I(v) \qquad z_{vq}^i \leq \chi_q(\mathbf{y}^i \odot \mathbf{x}^i) \tag{9}$$

where $\mathbf{y}^i \odot \mathbf{x}^i = (y_{v_1k_1}^i x_{v_1}^i, y_{v_1k_2}^i x_{v_1}^i, \ldots, y_{v_2k_1}^i x_{v_2}^i, \ldots)$, with $v \notin \{v_1, v_2, \ldots\}$.

Observe that composition and interface functions address more general compatibility patterns than *diversion feasibility*, *simultaneity*, *capacity* constraints (see [8]) and AND/OR conditions between submodules. Indeed, the latter can be easily expressed as linear constraints on $\mathbf{x}^i$.

5.2.3 Commonality evaluation

In our framework we consider both component and attribute commonalities. A component is common if it is always allocated with the same attribute configuration across the variants of the product family, whereas a component attribute is common for a pair of variants allocating the relevant component if it takes the same value. Component and attribute commonalities are

modeled by the following constraints.

$$\forall i \in P, v \in L, q \in I(v) \qquad z^i_{vq} \leq r^i_q \tag{10}$$

$$\forall i < j \in P, q \in Q, k \in A(q) \qquad |y^i_{qk} - y^j_{qk}| \leq M(1 - w^{ij}_{qk}) \tag{11}$$

$$\forall i < j \in P, q \in Q, k \in A(q) \qquad w^{ij}_{qk} \leq r^i_q \tag{12}$$

$$\forall i < j \in P, q \in Q, k \in A(q) \qquad w^{ij}_{qk} \leq r^j_q \tag{13}$$

$$\forall q \in Q \qquad |A(q)| \qquad \left(\sum_{i=1}^{|P|} r^i_q - 1\right) - \sum_{i=1}^{|P|-1} \sum_{k \in A(q)} w^{i,i+1}_{qk} \leq M(1 - p_q) \tag{14}$$

Constraints (10)-(14) model implications. Constraints (10) are needed since the sets $I(v)$, $v \in L$, in general define a cover of Q, i.e., for each product a component can be allocated to more than one module. The allocation of component q to both the products i and j, and a different value assignment of attribute k imply $w^{ij}_{qk} = 0$, see constraints (11)-(13). In constraints (14) the term $\alpha = \sum_{i=1}^{|P|-1} \sum_{k \in A(q)} w^{i,i+1}_{qk}$ counts, for each relevant attribute of component k, the number of variants across the product family which share the same value. Such number cannot be greater than $|A(q)|$ times the number of variants that allocate component k, i.e., the term $\beta = |A(q)| \left(\sum_{i=1}^{|P|} r^i_q - 1\right)$. Clearly, $\beta - \alpha > 0$ implies $p_q = 0$.

Finally, observe that non-linear constraints (11) can be easily linearized, see [16].

5.3 Objective functions

Several objective functions and platform evaluation metrics have been proposed in the literature. Aspects as customer needs, engineering performances, product robustness, component commonalization and production cost are the most addressed topics, each of them concerning the problem from a different perspective. Actually the most important and ultimate objective is product family net present value (NPV). We show here the flexibility and generality of the proposed framework by expressing some meaningful objective functions and evaluation metrics in terms of the decision variables of the model.

From the market domain perspective, where the expectations of market segments and the behavior of customers are taken into account, the main focus is on the customer satisfaction. A suitable measure is how well the customer needs are met by the platform [17].

The following maximizes conformance to customer needs.

$$\max \frac{1}{|P|} \sum_{i \in P} \sum_{h \in F} \frac{\nu_h^i(\psi_h(y_0^i))}{|F|}, \tag{15}$$

where $\nu_h^i : \lambda_h^i \to \mathbb{R}_+$ (for $i \in P, h \in F$) is a market specific value-generating functions which translate the value of the h-th functionality of the product i into an absolute scalar quantity called "value" and usually expressed in monetary units.

From the engineering domain perspective, robustness and product/process commonality are the most important platforming objectives. The former allows flexibility and make easier the adaptation to changes. The latter pursues economies of scale: sets of common features, components and subassemblies in general lead to lowering production costs. The robustness of a product architecture can be controlled by limiting the use of extreme values for component attributes, since they could be difficult to meet and could cause poor performance. This is implemented in the following objective function.

$$\max \sum_{i \in P} \sum_{q \in Q} \sum_{k \in A(q)} r_q^i \frac{\sqrt{(y_{qk}^i - \underline{\vartheta}_k^q)(\bar{\vartheta}_k^q - y_{qk}^i)}}{\bar{\vartheta}_k^q - \underline{\vartheta}_k^q}. \tag{16}$$

Product commonality can be quantified by directly resorting to the variables w and p. The following objective maximizes attribute commonality:

$$\max \sum_{i=1}^{|P|-1} \sum_{j=i+1}^{|P|} \sum_{q \in Q} \sum_{k \in A(q)} w_{qk}^{ij}, \tag{17}$$

and the following maximizes component commonality:

$$\max \sum_{q \in Q} p_q. \tag{18}$$

Although the objectives (17) and (18) give a measure of commonality, they do not take into account the production cost savings due to the economy of scale. To this aim and similarly to [9], we can introduce a multiperiod production cost function which models production horizon and learning effect due to the production volumes.

$$\min \sum_{\tau \in T} \sum_{q \in Q} \sum_{i \in P} r_q^i (1 - p_q) C_q \zeta(n_i) + \sum_{\tau \in T} \sum_{q \in Q} p_q C_q \zeta \left(\sum_{i \in P} r_q^i n_i \right), \tag{19}$$

where:

- T is the number of periods in the production planning horizon;
- n_i is the overall production volume (i.e. number of parts) of the i-th variant;
- C_q is the unit production cost of physical component q;
- $\zeta : \mathbb{R} \to \mathbb{R}$ is a function representing learning effect:

$$\zeta(n) = \frac{n}{T}\left(1 - e^{\frac{T}{n\tau}}\right).$$

Finally, we observe that a meaningful and more suitable objective function can be obtained by considering a weighted sum of two or more of the previous objective functions, with a view of balancing a trade-off between market and engineering needs.

6 Computational issues

[To be filled]

7 Conclusion

In this paper a modelling framework for integrated module-based and scale-based platforming has been introduced. The mathematical model works on an extended product family architecture and uses native product platform design decision variables and abstract functions to address module composition and interface implementation.

[to be completed]

References

[1] Object-process methodology (opm).
URL `http://www.objectprocess.org`.

[2] O.L. de Weck. Determining product platform extent. In T.W. Simpson, Z. Siddique, and J.R. Jiao, editors, *Product Platform and Product Family Design - Methods and Applications*. Springer, New York, USA, 2006.

[3] O.L. de Weck, E.S. Suh, and D. Chang. Product family and platform portfolio optimization. In *ASME Design Engineering Technical Conference*, 2003. Chicago, IL, September 2-6, Paper number DETC2003/DAC-48721.

[4] B. D'Souza and T.W. Simpson. A genetic algorithm based method for product family design optimization. *Engineering Optimization*, 35(1):1–18, 2003.

[5] X. Du, J. Jiao, and M.M. Tseng. Product family modeling and design support: an approach based on graph rewriting systems. *AIEDAM*, 16(2):103–119, 2002.

[6] D. Frey, J. Palladino, J. Sullivan, and M. Atherton. Part count and design of robust systems. *Systems Engineering*, 10(3):203–219, 2007.

[7] K. Fujita. Product variety optimization under modular architecture. *Computer-Aided Design*, 34:953–965, 2002.

[8] K. Fujita, H. Sakaguchi, and S. Akagi. Product variety deployment and its optimization under modular architecture and module commonalization. In *1999 ASME Design Engineering Technical Conferences*, 1999. Paper number DETC99/DFM-8923.

[9] K. Fujita and H. Yoshida. Product variety optimization: simultaneous optimization of module combination and module attributes. In *2001 ASME Design Engineering Technical Conferences*, 2001. Paper number DETC2001/DAC-21058.

[10] H.M.H. Hegge and J.C. Wortmann. Generic bill-of-material: a new product model. *International Journal of Production Economics*, 23:117–128, 1991.

[11] G. Hernandez, J. K. Allen, and F. Mistree. A theory and method for combining multiple approaches for product customization. *International Journal of Mass Customisation*, 1(2).

[12] G. Hernandez, J. K. Allen, and F. Mistree. Platform design for customizable products as a problem of access in a geometric space. *Journal of Engineering Optimization*, 35(3):229–254, 2003.

[13] J. Jiao, T.W. Simpson, and Z. Siddique. Product family design and platform-based product development: A state-of-the-art review. *Journal of Intelligent Manufacturing*, 18:5–29, 2007.

[14] R. Kumar and V. Allada. Customer need driven function-behavior. platform formation. In *IDETC/CIE 2005 Design Automation Conference*, 2005. Long Beach, California, September 24-28.

[15] R. Kumar, V. Allada, and S. Ramakrishan. Ant colony optimization method for product platform formation. In *ASME Design Engineering Technical Conferences, Advances in Design Automation, and Computers and Information in Engineering Conferences*, 2004. Salt Lake City, Utah, September 28 - October 2, Paper number DETC2000/DAC-14264.

[16] L. Liberti. Reformulation techniques in mathematical programming, November 2007. Thèse d'Habilitation à Diriger des Recherches.

[17] Hölttä-Otto, K. and Otto, K. Platform concept evaluation. In T.W. Simpson, Z. Siddique, and J.R. Jiao, editors, *Product Platform and Product Family Design - Methods and Applications*. Springer, New York, USA, 2006.

[18] A. Messac, M.P. Martinez, and T.W. Simpson. A penalty function for product family design using physical programming. *ASME Journal of Mechanical Design*, 124:164–172, 2002.

[19] M. Meyer and A. Lehnerd. *The Power of Product Platforms*. The Free Press, New York, USA, 1997.

[20] W.L. Moore, J.J. Louviere, and R. Verma. Using conjoint analysis to help design product platforms. *Journal of Product Innovation Management*, 16(1):27–39, 1999.

[21] R. Rai and V. Allada. Modular product family design: Agent-based pareto optimization and quality loss function-based post-optimal analysis. *International Journal Production Research*, 41(17):4075–4098, 2003.

[22] Z. Siddique. Assembly process selection to minimize existing assembly system modification cost during new product family member design. In *Proceeding of International Design Engineering Technical Conferences and Computers and Information in Engineering Conferences*, 2005. Long beach, California, Paper number DETC2005-85016.

[23] T. W. Simpson. Product platform design and customization: Status and promise. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 18(1):3–20, 2004.

[24] T.W. Simpson, Z. Siddique, and J.R. Jiao. *Product Platform and Product Family Design - Methods and Applications*. Springer, New York, USA, 2006.

[25] M.A. Slevinsky and P. Gu. Modular platform design using mechanical bus architectures. *International Journal of Mass Customization*, 1(1):65–82, 2005.

[26] K. Ulrich. The role of product architecture in the manufacturing firm. *Research Policy*, 24:419–440, 1995.