Artificial Intelligence for Engineering Design, Analysis and Manufacturing (2002), *16*, 103–120. Printed in the USA.

DOI: 10.1017.S0890060402020097

Product family modeling and design support: An approach based on graph rewriting systems

XUEHONG DU,[1] JIANXIN JIAO,[2] and MITCHELL M. TSENG[3]

[1]Artesyn Technologies Asia-Pacific Ltd, 13-15 Shing Wan Road, Tai Wai, Shatin, N.T., Hong Kong
[2]School of Mechanical and Production Engineering, Nanyang Technological University, Singapore 639798
[3]Department of Industrial Engineering & Engineering Management, Hong Kong University of Science & Technology, Clear Water Bay, Kowloon, Hong Kong

(Received February 2, 2001; Accepted January 9, 2002)

Abstract

Earlier research on product family design (PFD) often highlights isolated and successful empirical studies with a limited attempt to explore the modeling and design support issues surrounding this economically important class of engineering design problems. This paper proposes a graph rewriting system to organize product family data according to the underpinning logic and to model product derivation mechanisms for PFD. It represents the structural and behavioral aspects of product families as family graphs and related graph operations, respectively. The derivation of product variants becomes a graph rewriting process, in which family graphs are transformed to variant graphs by applying appropriate graph rewriting rules. The system is developed in the language of programmed graph rewriting systems or PROGRES, which supports the specification of hierarchical graph schema and parametric rewriting rules. A meta model is defined for family graphs to factor out those entities common to all product families. A generic model is defined to describe all specific entities relevant to particular families. An instance model describes all product variants for individual customer orders. A prototype of a graph-based PFD system for office chairs is also developed. The system can provide an interactive environment for customers to make choices among product offerings. It also facilitates design automation of product families and enhances interactions and negotiations among sales, design, and manufacturing.

Keywords: Design Automation; Graph Grammar; Mass Customization Systems; Product Family; PROGRES

1. INTRODUCTION

Facing the buyers' market, many industries are now geared toward mass customization, which is the mass production of individually customized goods and services (Pine, 1993). A number of on-line mass customization systems have been launched recently that allow customers to express their needs, configure their personalized products with desired features, and order and track the order status through the internet (e.g., Cannondale, 2000; Customatix, 2000; Dell, 2000; Eyeplanet, 2000; Haworth, 2000; IDtown, 2000; Volvo, 2000). In fact, e-commerce becomes a major driving force, as well as important enabler, for shaping the future roadmap of mass customization (Economist, 2000; ThinkCustom, 2000).

In terms of product realization, the major challenge of mass customization lies in how to achieve an increasing variety for catering to customization while keeping low costs of variety fulfillment, which seems to be an oxymoron. In practice, developing product families has been recognized as an effective means to achieve the economy of scale in order to satisfy diverse customer needs (Meyer & Utterback, 1993). In addition to leveraging the cost of delivering variety, product family design (PFD) can reduce development risks by reusing proven elements in a firm's activities and offerings (Sawhney, 1998).

Earlier research on PFD often highlights isolated and successful empirical studies, for example, Lutron lighting systems (Spira, 1993), Nippondenso bicycles (Whitney, 1993), Swatch watches (Ulrich & Eppinger, 1995), Dell

Reprint requests to: Dr. Jianxin Jiao, School of Mechanical and Production Engineering, Nanyang Technological University, Nanyang Avenue 50, Singapore 639798. E-mail: mjiao@ntu.edu.sg

computers (Schonfeld, 1998), and Sony Walkmans (Sanderson & Uzumeri, 1995; Kota & Sethuraman, 1998). There was a limited attempt to explore the modeling and design support issues surrounding this economically important class of engineering design problems. Meyer and Lehnerd (1997) and Robertson and Ulrich (1998) emphasize the development of product platforms by extracting those common product elements, features, and/or subsystems that are stable and well understood, so as to provide a basis for introducing value-added differentiating features for a range of products. PFD is tackled by exploiting the shared logic of a product design so that the design can be "stretched" and/or "scaled" (Rothwell and Gardiner, 1990) or be "robust" (Chen et al., 1996; Simpson et al., 1996, 1999) in response to various requirements from the market. Fujita et al. (1998) study how to optimize the system structure and the configuration of a product family. Assuming the rationale of modular product architectures and component sharing (Ulrich, 1995), most research focuses on the architecture of product families without explicitly addressing the underlying logic of the PFD process.

Most mass customization systems allow customers to select product features and/or options, rather than providing a limited number of products. The desired product is configured when a set of customer-selected options is received (Baldwin & Chung, 1995). This may result in a problem: if options continue to be added, the number of variants grows exponentially. Effective handling of variants is thus imperative for PFD and subsequent production planning and control in order to reduce data redundancy (Wortmann & Erens, 1995). In addition, customers may be confused when facing miscellaneous choices (Huffman & Kahn, 1998). This requires an appropriate representation of the product family data, as well as modeling of product configuration.

As a well-accepted simple product model, the bill of materials (BOM) encodes the relationships between finished products and their constituent parts or assemblies. Some research has generalized BOM to include product families, such as the generic BOM (GBOM) concept (Hegge & Wortmann, 1991). While a GBOM aims at exploring a generic product structure and excels in describing manufacturing or production-related product information, its standpoint mainly rests on the assembly structure of a product family, which seems to ignore the design process prior to production. McKay et al. (1996) try to describe product families from both sales (customer) and assembly views through combining the GBOM with specific product modeling approaches. Their work however focuses on the physical structures of products only. Other methods include generic product modeling (Erens & Verhulst, 1997), the generic variety structure (Jiao et al., 2000), and the data model introduced by Baldwin and Chung (1995).

There is vast literature devoted to configuration design (Brown & Birmingham, 1997; Darr et al., 1998). Many approaches have been reported such as logic-based approaches, resource-based approaches, constraint-based approaches, and case-based reasoning approaches (Sabin & Weigel, 1998). Most modeling work deals with design synthesis, which means how a valid design is assembled from instances of a fixed set of predefined component types (Mittal & Frayman, 1989). While configuration design focuses on individual products only, PFD involves a family of products where product to product and product to family relationships become important concerns, as well as the representation of variety and the architecture of product families (Du et al., 2001). Jiao and Tseng (1999) discuss the information modeling of PFD. It is pointed out that, in addition to the representation of variety, understanding the underlying logic of PFD and modeling the product variant derivation process are of primary importance. In addition, existing modeling approaches usually define product families in terms of a single perspective (mostly the structural view, e.g., maintenance) and for a specific purpose (e.g., operations planning; Van Houten et al., 1998). To deal with variety, which encompasses the entire product development process, it is necessary to represent product families for different business functions, including sales and marketing (functional) and engineering (structural) views.

Toward this end, we propose a graph rewriting system that organizes product family data according to the underpinning logic among them and models product configuration mechanisms to support PFD. The system is formally defined in a high-level, multiparadigm specification language, PROGRES, which combines concepts from programmed graph rewriting systems and supports the specification of hierarchical graph schema and parametric rewriting rules (Schürr, 1994, Schürr et al., 1998). The general process of PFD and its key issues are introduced in Section 2. Accordingly, the constructs of the graph rewriting system for PFD are discussed in Section 3, along with guidelines of graph grammar-based PFD modeling. Sections 4 and 5 describe the graph schema and graph transformations for PFD modeling. A case study of office chairs is presented in Section 6 to illustrate the feasibility and potential of the proposed approach. Discussion takes place and conclusions are drawn in Section 7.

2. PFD DESCRIPTION

A product family refers to a group of individual products that share common subsystems or components and yet possess specific functional features to satisfy a variety of market niches (Meyer & Lehnerd, 1997). In practice, different business departments tend to interpret and employ product families in different ways. From the marketing and sales perspective, product families exhibit the company's product line or product portfolio and thus are characterized by various sets of functional features for diverse customer groups. The engineering view of product families embodies product technologies and associated manufacturability and

is thereby characterized by differences in product structures, design parameters, and components. The synchronization of multiple views is an important issue of PFD modeling (Jiao & Tseng, 1999; Du et al., 2001).

The general process of PFD is illustrated in Figure 1. From the viewpoint of sales, a product family can be specified in terms of three elements: common features, distinctive features, and selection constraints. Common features indicate the similarity of customers' requirements related to a particular market segment. A few optional values are selectable for each distinctive feature in order to satisfy specific customer requirements. Selection constraints are defined for presenting customers with only feasible options.

From the engineering perspective, PFD is characterized by a generic product structure (GPS; Du et al., 2001). As a generic data structure of product families, the GPS is a hierarchy comprising modules and their interrelationships at different levels of abstraction. Two types of modules are distinguished: primitive modules and compound modules. The first type refers to those modules that cannot be further decomposed. Each of them possesses several variants. The second type is composed of common modules, primitive modules, and/or other compound modules. This means that a compound module may be used as a child module to compose another compound module (i.e., the parent module). In this sense, the end product itself is in fact a compound module. The variants of a compound module result from variations of its child modules. In addition, modules are associated with variety parameters. Variety parameters originate from distinctive functional features and propagate along the hierarchy of a GPS. The value of a variety parameter associated with a child module is determined by the variety parameters of its parent module. Furthermore, inclusion conditions can be defined in terms of variety parameters to specify the circumstances under which a module is a constituent of another module or a particular variant becomes the instance of a primitive module.

With the above understanding, a custom product design involves two phases: customer selection in the sales view and product variant derivation in the engineering view. Customers make their selections among sets of options. The selected functional features and their values represent

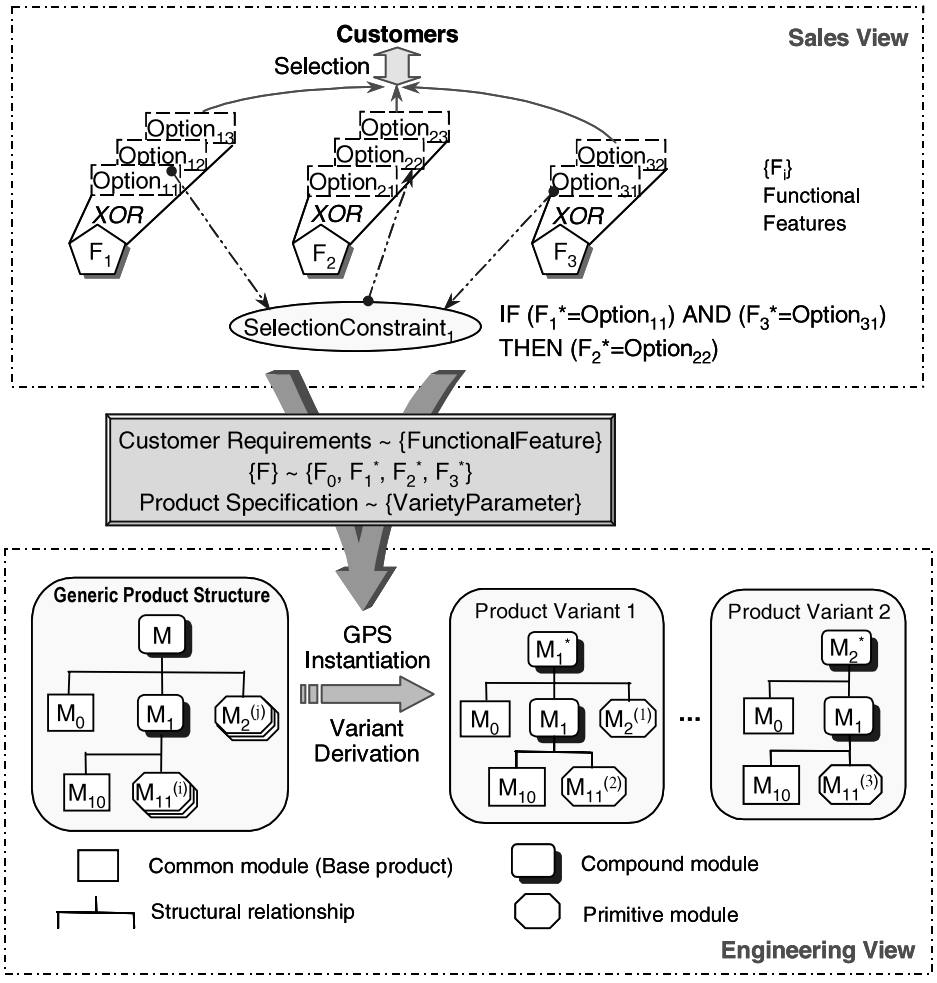


Fig. 1. The general process of product family design.

the customer requirements, thus becoming the content of the product specification. The variety parameters of the end product are defined according to the product specification and then propagate to its constituent modules. Each item in the GPS (either a module or a structural relationship) is instantiated according to associated inclusion conditions. Therefore, while a GPS characterizes a product family, each instance of GPS corresponds to a product variant of the family. Propagation of variety parameters and inclusion conditions embody the configuration mechanisms of product variant derivation in PFD.

3. GRAPH REWRITING SYSTEMS FOR PFD

Graph grammars are widely used as a tool to model complex systems. Du (2000) proposes a graph grammar based approach to the formal representation of product families. PFD can be described in programmed attributed graph grammars, in which features and modules are represented as nodes, relationships among features and modules are represented as edges, and manipulations of modules are modeled as productions. The graph representing items that construct the starting point of the family design is used as a starting graph. The graph of a product variant is transformed from the starting graph by invoking proper productions. A product family is thus defined by all graphs that can be generated from the starting graph according to predefined productions and the control diagram, which form the graph language of the family.

Graph grammar formalisms can be implemented as a graph rewriting system. The development of a programmed graph rewriting system involves two closely related subtasks. One is to design a graph model for the corresponding complex object structure, namely, the graph schema. It is a set of graph entities common to a certain class of graphs and can describe all necessary types of nodes and edges, as well as their associated attributes and static integrity constraints. The other is to program object (graph) operations for analysis and modification by composing and sequencing subgraph tests and graph rewriting rules, namely, graph transformations.

Graph rewriting systems can be formalized by writing a PROGRES specification. PROGRES and its programming environment are based on the data model of directed attributed graphs and offer a concept of programmed graph rewriting systems to describe complex graph transformations. PROGRES specifications determine both the static structure and the allowed dynamic behavior of a system. It excels in handling derived fact (attribute dependence), providing parametric rewriting rule specification, and supporting data and system consistency (Schürr et al., 1995, 1998).

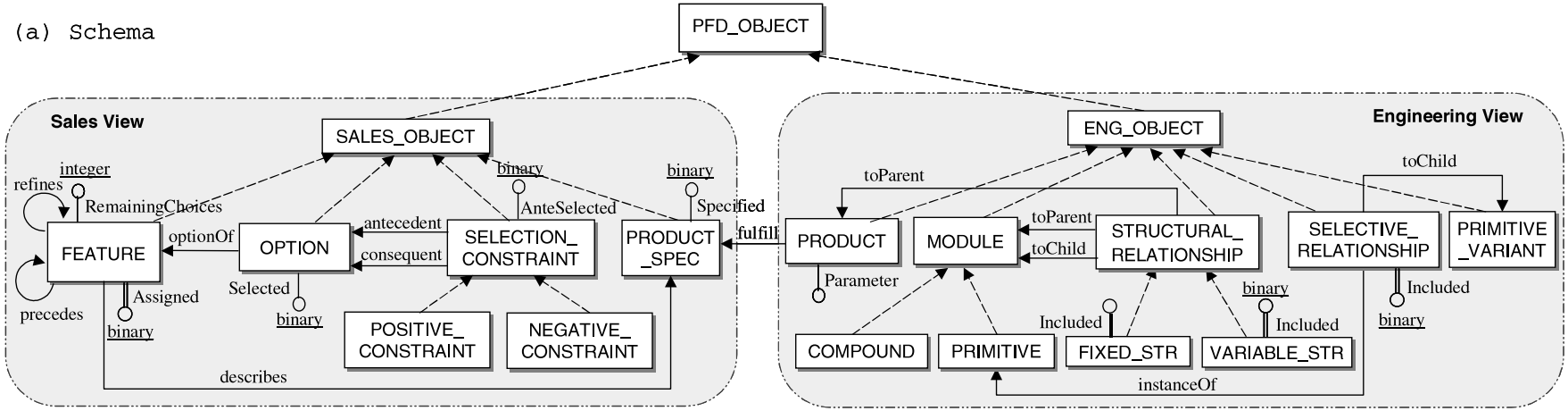
Because PROGRES is a stratified language, the graph rewriting system for PFD involves three layers: the meta, generic, and instance models. A generic model represents the generic data structure of a product family, in which family-related items such as features and modules are specified using node classes. The graphical presentation of a generic model is called a family graph. The abstraction of those common graph entities in specific generic models for different product families are specified by defining node classes and edge types, which compose a meta model. To generalize those graph operations common to all product families, the related graph transformations are defined at the meta level. For a particular product family, a generic model can be defined by adapting the relevant entities of the meta model to specific characteristics of the product family. A family graph can be instantiated to a variant graph, thus called an *instance model*, representing the modules and structure of a specific product variant. The instance model is composed of the *node instance* together with the edges. Because PROGRES is a strongly typed language, the attributes defined in the classes can be inherited to type-level entities and the attributes defined in the types can be inherited to instance-level entities.

The generic approach to PFD modeling based on graph rewriting systems involves the construction and application phases (for a general understanding of the PROGRES specification of graph rewriting systems, please refer to fig. 7 in Heimann et al., 1995). The *meta model* and graph transformations are defined by the classes and productions in the PROGRES language. To adapt the meta model to a family-specific generic model, we specify the node types of the product family from two different views: sales and engineering. By merging these specific types with the classes and productions, we obtain a complete graph grammar specification of the particular product family. The starting graph for sales and marketing consists of features, options, product specifications, and selection constraints, as well as the relationships among them, and is thus called the sales view family graph. The starting graph for engineering consists of generic modules, primitive variants, and generic structural or selective relationships associated with the modules and variants; it is called the engineering view family graph. During the application stage, sales persons and customers input their options of features. The sales view family graph is then rewritten according to the control structure predefined in the construction stage. The result will be a variant graph–graphical representation of a product specification. This specification is then transferred to the engineering view in the form of variety parameters of the end product. Taking these variety parameters as input, the engineering view family graph starts to transform according to the control structure defined in the meta model. The resulting variant graph (i.e., the instance model) represents the definition of an order-specific product.

4. META MODEL AND GRAPH TRANSFORMATION

Specifying those entities common to all product families, a meta model consists of the class-level graph schema (node classes and edge types) and graph transformations. Figure 2

(a) Schema



(b) Operations

transaction SpecifyProduct [1:1] = … end

transaction DefProduct [1:1] (aProduct : PRODUCT_) = … end

production AssignValue (aFeature : FEATURE, anOption : OPTION) = … end

production ProcessPositiveConstraint (aFeature : FEATURE) = … end

production ProcessNegaitiveConstraint (aFeature : FEATURE) = …end

production TransferSpec (aSpec : PRODUCT_SPEC, aProduct : PRODUCT) =… end

production GetFeatureValue (aSpec : PRODUCT_SPEC, aProduct : PRODUCT, i: integer) = =… end

production RemoveNotIncludedVariant (aPrimitive : PRIMITIVE) = … end

production RemoveNotIncludedModule (aModule : MODULE) =… end

production CloseSpec (aProductSpec : PRODUCT_SPEC) =… end

production = BuildUpParentChildRelation =… end

test FindUnAssignedFeature (aProduct : PRODUCT_SPEC, out aFeature) [0:1] =… end

test ConfirmNoUnassignedPrecedant (aFeature : FEATURE [0:1] =… end

test ConfirmNoUnassignedRefinee (aFeature : FEATURE) [0:1] =… end

path toConsequentP : OPTION -> OPTION =… end

path toConsequentN : OPTION -> OPTION =… end

path toFeatureValue : PRODUCT -> FEATURE_VALUE (aFeature : FEATURE) =… end

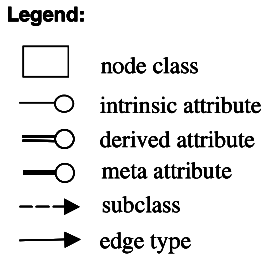


Fig. 2. The graph grammar specification of PFD (meta models and graph transformations).

summarizes the graph grammar specification of the PFD at the meta level.

4.1. Class-level graph schema

The class level of the PROGRES graph schema factors out all common entities of product families. All classes of nodes and all types of edges occurring in PFD systems are defined. PFD_OBJECT acts as the root of the class hierarchy, as shown in Figure 2(a). Two subclasses can be distinguished, SALES_OBJECT and ENGINEERING_OBJECT, which are elaborated in the sales view meta model and the engineering view meta model, respectively. Figure 2(a) gives the graphical representation of node classes and edge types in both sales and engineering views, as well as the inheritance relations.

As shown in Figure 2(a), SALES_OBJECT is a superclass that covers all entities occurring in the sales view meta model. An optionOf edge between the FEATURE and OPTION nodes models the fact that one or more options can be selected to define a feature.

To handle selection constraints, we connect the antecedent of a selection constraint to its consequent via SELECTION_CONSTRAINT nodes. The derived attribute, AnteSelected, of SELECTION_CONSTRAINT nodes is determined by the Selected attribute of its antecedent. A SELECTION_CONSTRAINT can be a POSITIVE_CONSTRAINT or a NEGATIVE_CONSTRAINT. A *positive constraint* is defined as follows: if a specific option A^k of feature A (the antecedent) is selected, then feature B must take on a specific option B^l (the consequent), where the superscript k or l, denotes a particular option (i.e., a specific value) of the feature. A *negative constraint* means that if an option A^k of feature A (the antecedent) is selected, then a feature B must not take on option B^l (the consequent).

A precedes edge models the fact that a feature whose option is the antecedent(s) of the option(s) of another feature must be specified prior to the specification of the latter feature. An example of the refines relationship is that feature StyleOfArmrests refines feature WithArmrests. PRODUCT_SPEC is linked to each feature node that describes the feature from certain aspects. The remaining elements used to declare node attributes will be introduced in Section 5. Assigned is a derived attribute of FEATURE, indicating whether the value of this feature has been assigned (true) or not (false). Its default value is false. When an eligible feature value is selected, it becomes true. The value of RemainingChoices is recalculated by transformations associated with FEATURE. On the other hand, an edge-type specification defines what node types are admissible at the end points of the edge; for example, an optionOf edge starts from OPTION and ends at FEATURE.

As shown in Figure 2(a), ENG_OBJECT is a superclass that covers all entities occurring in the engineering view meta model. A MODULE can be a PRIMITIVE or a COMPOUND one. A primitive module cannot be further decomposed and can be realized by one or more PRIMITIVE_VARIANTs. A compound module consists of some lower level modules that may be primitive or compound ones. PRODUCT comprises modules of different kinds.

Another class of ENG_OBJECT is STRUCTURAL_RELATIONSHIP. One of its subclasses FIXED_STR, models those fixed structural relationships between parent and child modules. To indicate a FIXED_STR, an attribute, Included, is introduced as a meta attribute whose value equals true. Another subclass is VARIABLE_STR, which models structural variations, that is, a child module may or may not be included in the parent module. The derived attribute Included is determined by parameters of the parent module. Another node class is SELECTIVE_RELATIONSHIP. It models the relationship between a primitive module and its variants. The Included attribute of a SELECTIVE_RELATIONSHIP node is instantiated according to the parameters of primitive modules. Included = true means that a primitive variant is included in the resulted product structure.

Two types of edges are specified in the engineering view as shown in Figure 2(a): toParent and toChild. A STRUCTURAL_RELATIONSHIP node is linked to the parent module by a toParent edge and to the child module by a toChild edge. A SELECTIVE_RELATIONSHIP node is linked to a PRIMITIVE node by a toParent edge and to a PRIMITIVE_VARIANT node by a toChild edge.

The primary concern of defining node classes is to denote coercions of node types that possess common properties. As a result, the concepts of classification and specialization are introduced. This eliminates duplicating declarations by supporting *multiple inheritances* along the edges of the class hierarchy. In addition, node classes play the role of meta types. As types of node types, they support the controlled use of formal (node) type parameters within generic subgraph tests and graph rewriting rules. This is demonstrated in the following section.

4.2. Graph operations

Section 4.1 dealt with the static portion of PFD systems; this section discusses the modeling of the operational behavior. Complex operations are defined as transactions that are composed of a set of productions to be executed following a controlled sequence. These basic operations are defined as productions in PROGRES. By nature they are at the meta level and are independent of particular product families.

To support PFD, some operations are necessary at the meta level. First, there should be operations to let customers choose among available options. Second, if an option that is the antecedent of a selection constraint is selected, there should be operations to process the consequent according to whether it is a negative or a positive constraint. Third, there should be operations to transform the selected

options to variety parameters, that is the relevant attributes of the PRODUCT node in the engineering view family graph (to be elaborated in Section 5). Fourth, there should be operations to delete modules and primitive variants that are not included in the graph of a desired product variant. Figure 2(b) illustrates all the operations involved in the system.

Figure 3(a) depicts a production, AssignValue, designed for customers to assign values to certain features, which is realized by making choices among selectable options. The dashed rectangles above and below the separator ::= define the left-hand side (lhs) and right-hand side (rhs) of the production, respectively. The rule can be applied only if all conditions are fulfilled. The first statement in the condition part is used to check whether the feature has been assigned a value. The second and third conditions ensure that there is no FEATURE to which the value should be assigned before the value is assigned to this feature or that the values of all features preceding or being refined by this feature have been assigned. If all condition statements hold true, the elements of the lhs in the family graph are replaced by the elements of the rhs. Those unselected options are thus removed from the graph and all node attributes receive their new values according to the transfer functions.

Once an option is selected for a feature, it is necessary to check whether this selection affects the options of other features. If the selected option is an antecedent of a negative selection constraint, the consequent of the constraint

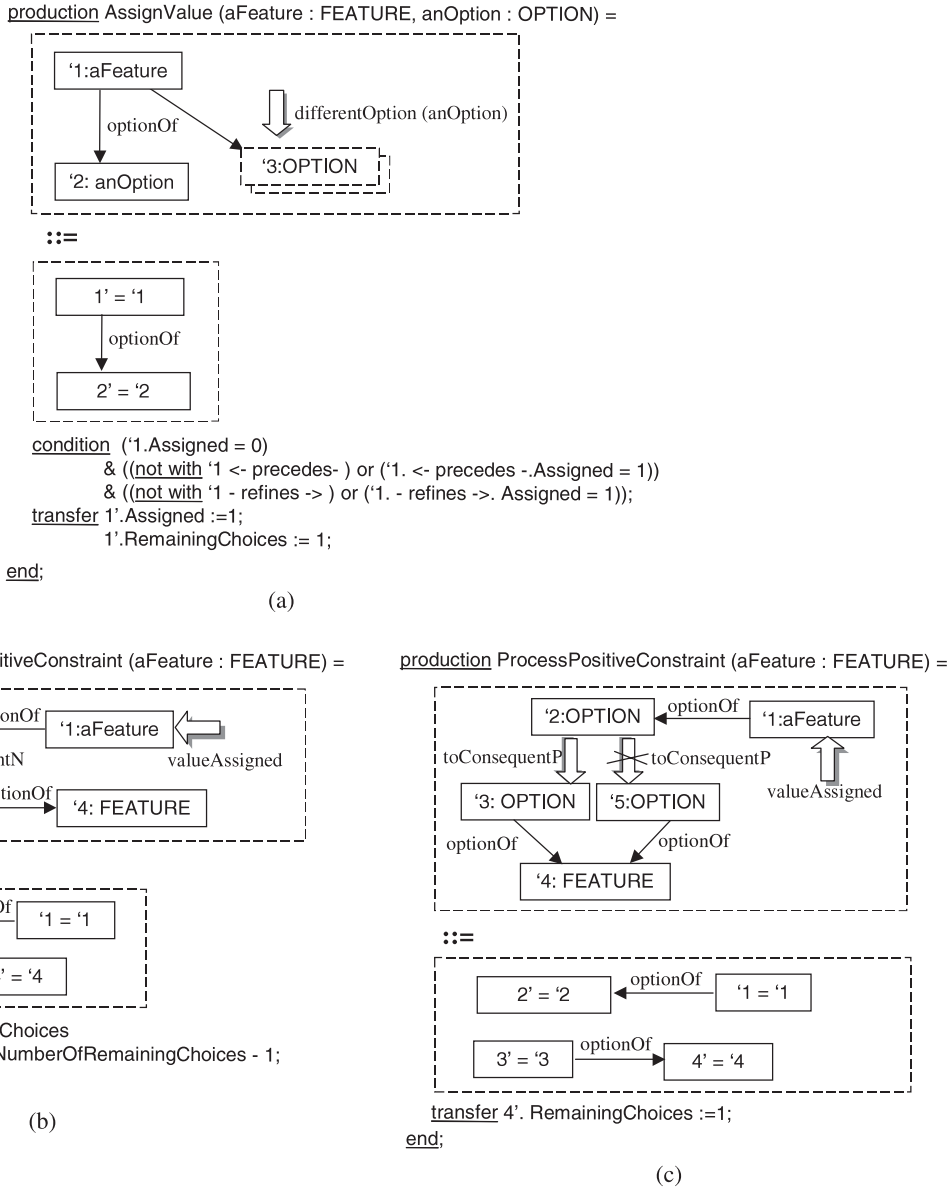


Fig. 3. The basic productions for the sales view meta model: (a) assigning values to a feature (selecting an option), (b) processing a negative constraint, and (c) processing positive constraint.

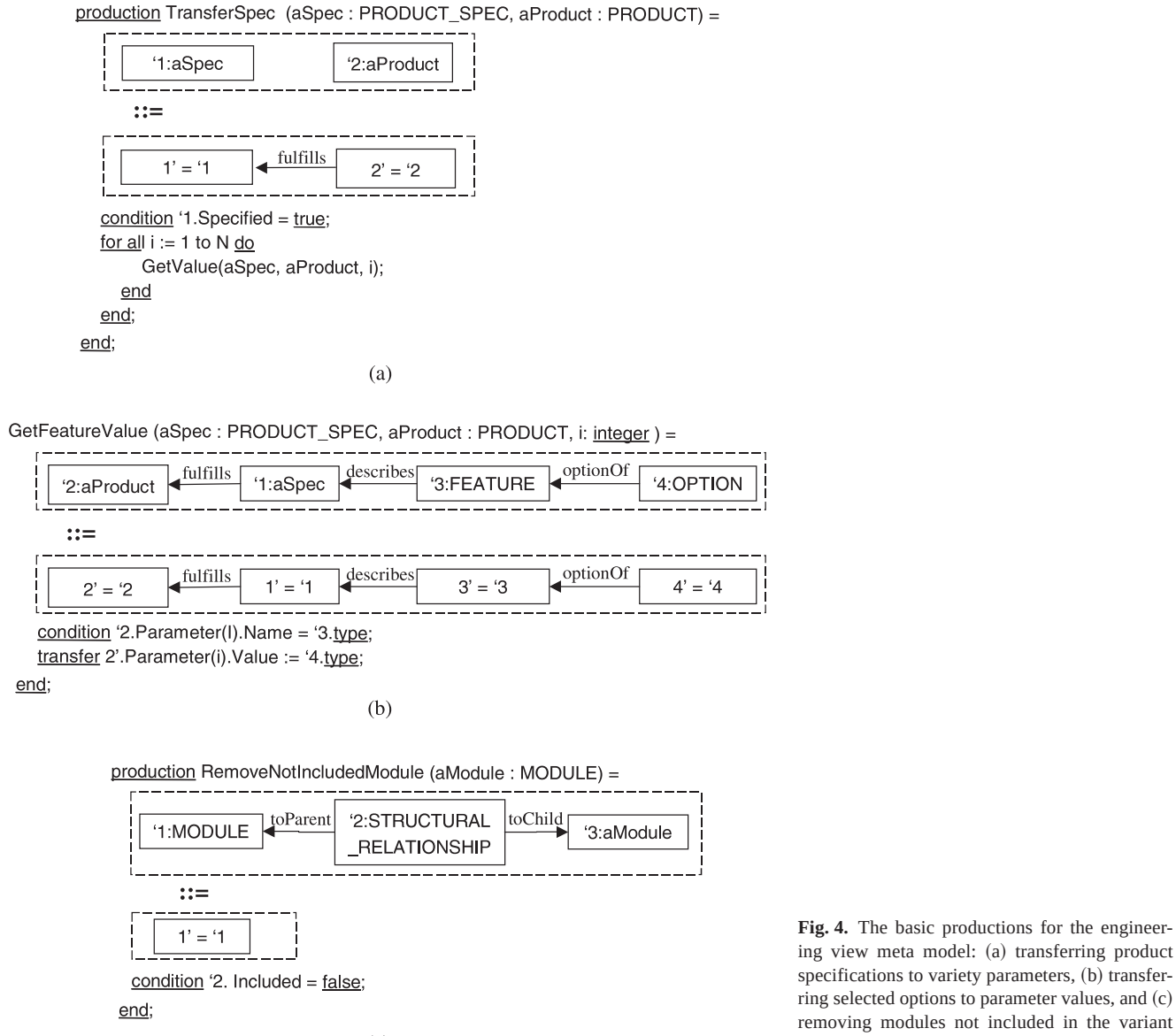


Fig. 4. The basic productions for the engineering view meta model: (a) transferring product specifications to variety parameters, (b) transferring selected options to parameter values, and (c) removing modules not included in the variant graph.

should be deleted from those selectable options of the affected feature. Figure 3(b) is a production designed for this purpose.[1] For aFeature (an instance of FEATURE) whose value has been assigned (indicated by the hollow arrow, valueAssigned, attached to the aFeature node), if there is a path from its selected option to the consequent of a negative constraint (indicated by the hollow arrow, toConsequentN, between two OPTION nodes), the consequent together with the constraint node will be removed. As a result, the RemainingChoices of the affected feature will be reduced by one. If the selected option is an antecedent of a positive selection constraint, the Selected attribute of its consequent should be assigned a new value of true and all other options will be deleted, as shown in Figure 3(c). The hallow arrow, toConsequentP, with a cross between nodes '2 and '5 leads to those unselected options of the affected feature. Consequently, the constraint node and unselected options are both removed and the RemainingChoices is changed to 1.

Figure 4(a) shows a production TransferSpec, which defines a rewriting rule to derive the engineering description of a product variant according to the sales specification. If a product is specified (stated as condition), a fulfill edge is added between the PRODUCT_SPEC node and the PRODUCT node. The N denotes the number of variety parameters. Figure 4b depicts a production GetFeature-

[1] A hollow fat arrow between two nodes requires the existence of a certain path (derived relationship) between these two nodes, whereas a hollow fat arrow attached to a single node requires that its target node fulfills a certain restriction (belongs to a derived node set).

Value, which is used to trace the OPTION nodes connected to the aSpec node. The options are transferred to the attribute values of the PRODUCT node, representing the values of variety parameters.

Once a product specification is transferred to variety parameters, all derived attributes in the engineering view family graph start to be evaluated automatically. If a module is not to be included by its parent module (i.e., the Included attribute of the associated STRUCTURAL_RELATIONSHIP node is false), the corresponding node of this child module will be deleted. A production RemoveNotIncludedModule is designed for this purpose, as shown in Figure 5(c). Similarly, if a variant is not to be instantiated for its primitive module, the Included attribute of the corresponding SELECTIVE_RELATIONSHIP node should take on false, and thus the associated PRIMITIVE_VARIANT node is removed.

In addition to these productions, other operations on family graphs can be defined in a similar way to facilitate PFD. Some examples are given in Figure 5.

4.3. Product variant derivation

The process of transforming a family graph to a variant graph is rather complex and may not be able to be specified using a single production. Imperative control structures can enforce certain orders of production application (Schürr, 1990) that are specified as transactions. Figure 6 depicts the control structures for product variant derivation. First of all, a graph test on the sales view family graph is performed, as shown in Figure 6(a). A feature whose values are not assigned is thus selected. If there is no value-unassigned feature preceding this selected feature or being refined by it, the selected feature and its selectable options are pre-

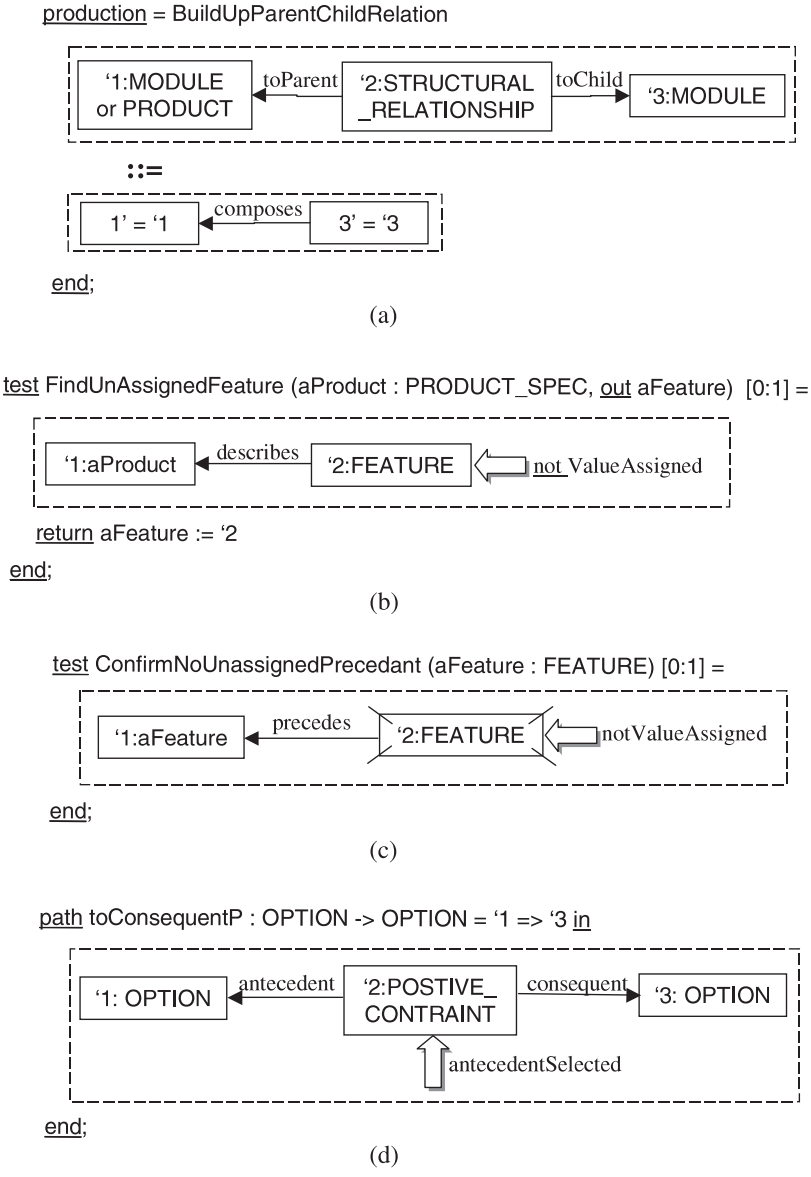


Fig. 5. Definitions of supplementary productions: (a) changing the structural relationship node to a composes edge, (b) a test of finding a value-unassigned feature, (c) a test of confirming no unassigned precedent, and (d) the path to the consequent of a positive selection constraint.

```
transaction SpecifyProduct [1:1]
      use aFeature :FEATURE do
          loop
                FindUnassignedFeature (out aFeature);
            & ConfirmNoUnassignedPrecedent (aFeature);
            & ConfirmNoUnassignedRefinee (aFeature);
            & write (aFeature, selectableOption);
            & read (anOption in selectableOption);
            & AssignFeatureValue (aFeature, anOption);
            & use aConstraint: NEGATIVE_CONSTRAINT do
                  loop
                     ProcessNegativeConstraint (aFeature, anOption);
                  end
              end [0:n];
            & use aConstraint: POSITIVE_CONSTRAINT do
                  loop
                     ProcessPositiveConstraint (aFeature, anOption);
                  end
              end [0:n];
          end
      end;
    & CloseSpec (aProductSpec);
end;
```

(a)

```
transaction DefProduct [1:1] ( aProduct : PRODUCT)
    TransferSpec (aSpec, aProduct);
  & use aModule: MODULE do
      loop
        RemoveNotIncludedModules (aModule);
      end
    end;
  & use aPrimitiveVariant: PRIMITIVE_VARIANT do
      loop
        RemoveNotIncludedPrimitiveVariants (aPrimitiveVariant);
      end
    end;
end.
```

(b)

Fig. 6. A specification of control structures: (a) deriving product variant specification (sales view) and (b) deriving product variant definition (engineering view).

sented to customers for them to choose. After obtaining the customers' input (in the form of a selected option), a production, AssignValue, is applied to the family graph. If the selected option is the antecedent of certain constraints, these constraints will be processed. Then we go back to process another value-unassigned feature till every feature obtains a value. When all features are assigned, the product specification process terminates and the attribute Specified of the PRODUCT_SPEC node becomes true.

Figure 6(b) shows the control structure for product definition. It seeks to transform the product specification described in sales terms to the product definition described in engineering terms. This operation is automatic because the parameter propagation from parent modules to child modules and inclusion conditions were modeled as derived attributes. In the first step, a production, Transferspec, is applied to transfer the product specification to variety parameters of the PRODUCT node. Derived attributes are revaluated. Then all those unincluded modules and primitive variants are removed. The desired variant graph is thus derived.

5. GENERIC MODEL AND FAMILY GRAPH

To support a specific PFD, meta models need to be adapted to particular product families, that is, to be transformed to generic models. To do so, all family-specific features, options, and generic items in the GPS are specified as node types. A node-type declaration defines the label of a group of nodes and the node class to which they belong. The purpose is to define the behavior of the nodes of this type, which indicates the functional dependencies of the attributes. Three kinds of attributes are identified: intrinsic, meta, and derived. Figure 7 gives an example of node-type definitions of Chair, Underframe, and Stand for an office chair product family.

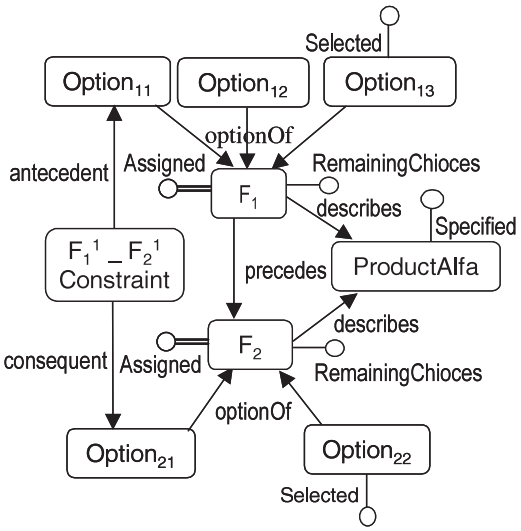
Those attributes whose values need to be input by customers can be assigned as intrinsic attributes that possess a value independent of the values of other attributes. For example, for an office chair product family such attributes as Color, WithArmrests, StyleOfArmrests, Turnable, and Drivable of the node Chair are intrinsic attributes. An intrinsic attribute has a type-dependent *initial value*, which may be changed directly by performing an appropriate graph transformation. If a parameter is modeled as an intrinsic attribute, its default value can be set to be the initial value of the attribute.

Those attributes whose values are common among family members can be assigned as meta attributes that possess constant type-dependent values. This enables the handling of those node properties having the same value for all instances of a given node type. For example, for the office chair product family a statement that the value of a meta attribute, Durable, is Yes implies that all product variants of this family are Durable.

Parameter propagation from a parent node to its child nodes can be modeled by derived attributes that have node instance-specific values and change their values as a result of performed graph transformations. All instance-specific values are determined by means of directed equations only. For example, for the office chair product family the derived

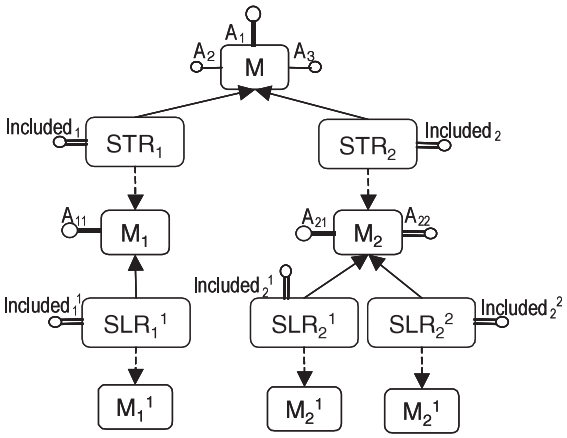
```
node_type Chair: PRODUCT
    intrinsic Color:= [undefined];
    intrinsic WithArmrests:= [undefined];
    intrinsic StyleOfArmrests := [undefined];
    intrinsic Turnable:= [undefined];
    intrinsic Drivable := [undefined];
    meta Durable := Yes;
end;
node_type UnderFrame: COMPOUND
    meta Durable := Yes;
    derived Turnable:= [self.-parent->.Turnable | false];
    derived Drivable:= [self.-parent->.Drivable | false];
end;
node_type Stand: PRIMITIVE
    derived Turnable:= [self.-parent->.Turnable | false];
end;
```

Fig. 7. Declarations of intrinsic, meta, and derived attributes.



Assigned$_1$:= [((Slected$_{11}$) AND (NOT(Slected$_{12}$)) AND (NOT(Slected$_{13}$))) OR (NOT(Slected$_{11}$) AND (Slected$_{12}$) AND (NOT(Slected$_{13}$))) (NOT(Slected$_{11}$) AND (NOT(Slected$_{12}$)) AND (Selected$_{13}$)|0]
NumberOfRemainingChoices$_1$:=3
Assigned$_2$:= [(Slected$_{21}$) AND (NOT(Slected$_{22}$)) OR (NOT(Slected$_{11}$) AND (Slected$_{12}$))|0]
NumberOfRemainingChoices$_2$:=2
Slected$_{11}$:= 0
Slected$_{12}$:= 0
Slected$_{13}$:= 1
Slected$_{21}$:= 0
Slected$_{22}$:= 1

(a)



Included$_1$:=*f*(self.–toParent->.A$_3$)
Included$_2$:=1
Included$_1^1$:= 1
Included$_2^1$:=*f*(self.–toParent->.A$_{22}$)
Included$_2^2$:=*f*(self.–toParent->.A$_{22}$)
A$_1$= A$_1^0$
A$_{11}$= A$_{11}^0$
A$_{21}$= A$_{21}^0$
A$_{22}$:= *f*(self.–parent->.A$_2$)

Legend:

node type — toParent — toChild — intrinsic attribute — derived attribute — meta attribute

(b)

node_type ProductAlfa: PRODUCT_SPEC;
end;
node_type F$_1$: FEATURE;
  derived Assigned := [((Selected$_{11}$) AND (NOT(Selected$_{12}$)) AND (NOT(Selected$_{13}$))) OR (NOT(Selected$_{11}$) AND (Selected$_{12}$) AND (NOT(Selected$_{13}$))) (NOT(Selected$_{11}$) AND (NOT(Selected$_{12}$)) AND (Selected$_{13}$)|**false**];
  intrinsic RemainingChoices:=3;
end;
node_type F$_1$: FEATURE;
  derived Assigned := [((Selected$_{11}$) AND (NOT(Selected$_{12}$)) OR (NOT(Selected$_{11}$) AND (Selected$_{12}$)) | **false**];
  intrinsic RemainingChoices:=2;
end;
node_type F$_1^1$-F$_2^1$Constraint: POSITIVE_CONSTRAINT;
end;
node_type Option$_{11}$, Option$_{12}$, Option$_{21}$: OPTION;
  Selected := **false**;
end;
node_type Option$_{13}$, Option$_{22}$: OPTION;
  Selected := **true**;
end;

(c)

node_type M: PRODUCT
  meta A$_1$:= A$_1^0$;
  intrinsic A$_2$:= [undefined];
  intrinsic A$_3$:= [undefined];
end;
node_type STR$_1$: FIXED_STR
end;
node_type STR$_2$: VARIABLE_STR
  Included := [*f*(**self.**-toParent->.A$_3$) | **false**];
end;
node_type SLR$_1^1$: SELECTIVE_REALTIONSHIP
  Included := **true**;
end;
node_type SLR$_2^1$, SLR$_2^1$: SELECTIVE _REALTIONSHIP
  Included := [*f*(**self.**-to parent->.A$_{22}$)| **false**];
end;
node_type M$_2$: PRIMITIVE
  meta A$_{21}$:= A$_{21}^0$;
  derived A$_{22}$:= *f*(**self.**-toParent->.A$_2$);
end;
node_type M$_1^1$, M$_2^1$, M$_2^2$: PRIMITIVE_VARIANT
end;

(d)

Fig. 8. (a) Sales view and (b) engineering view family graphs and node-type specifications for (c) sales view and (d) engineering view generic models for a specific product family.

Table 1. *Features, options, and selection constraints for the office chair product family*

	Feature	Option
Common feature	F_0: Durable	Yes
Optional feature	F_1: Color	Red, Blue, Gray
	F_2: Turnable	{Yes, No}
	F_3: Drivable	{Yes, No}
	F_4: WithArmrests	{Yes, No}
	F_5: StyleOfArmrests	{Plain, Deluxe}
Selection constraint	If Drivable = Yes, Turnable $\neq$ No;	
	If Color = Gray, WithArmrests = Yes AND StyleOfArmrests = Deluxe.	

```
node_type OfficeChair : PRODUCT_SPEC
end;
node_type Color : FEATURE
      derived Assigned := [/(self.<-optionOf- OPTION.Selected) |false];
      intrinsic RemainingChoices := 3;
end;
node_type Durable : FEATURE
      meta Assigned := true;
      meta RemainingChoices := 1;
end;
...
node_type C_Sconstraint : POSITIVE_CONSTRAINT
end;
node_type D_Tconstraint : NEGATIVE_CONSTRAINT
end;
```

(a)

```
node_type OfficeChair : PRODUCT
      Parameter.Color := [undefined];
      Parameter.WithArmrests := [undefined];
      Parameter.StyleOfArmrests := [undefined];
      Parameter.Turnable := [undefined];
      Parameter.Drivable := [undefined];
      meta Durable := Yes;
end;
node_type UnderFrame : COMPOUND
      meta Durable := Yes;
      Parameter.Turnable := [self.-composes->.Turnable | false];
      Parameter.Drivable := [self.-composes->.Drivable | false];
end;
...
node_type Stand : PRIMITIVE
      meta Durable := Yes;
      Parameter.Turnable := [self.-composes->.Turnable | false];
end;
...
node_type AR_a : PRIMITIVE_VARIANT
      Included := [(self.- toParent ->. WithArmrests = Yes) & (self. – toParent ->.
                  StyleOfArmrest = Plain) | false];
end;
...
node_type ChairArmrests : VARIABLE_STR
      Included := [self.-toParent->.WithArmrests | false];
end;
node_type ChairUnderframe, ChairBack, ChairSeat, UnderframeStand,
            UnderframeSupport, BackBackframe, BackBackupholstery, SeatSeatframe,
            SeatSeatupholstery : FIXED_STR
end;
```

(b)

Fig. 9. Node-type specifications for (a) sales view and (b) engineering view generic models for the office chair product family.

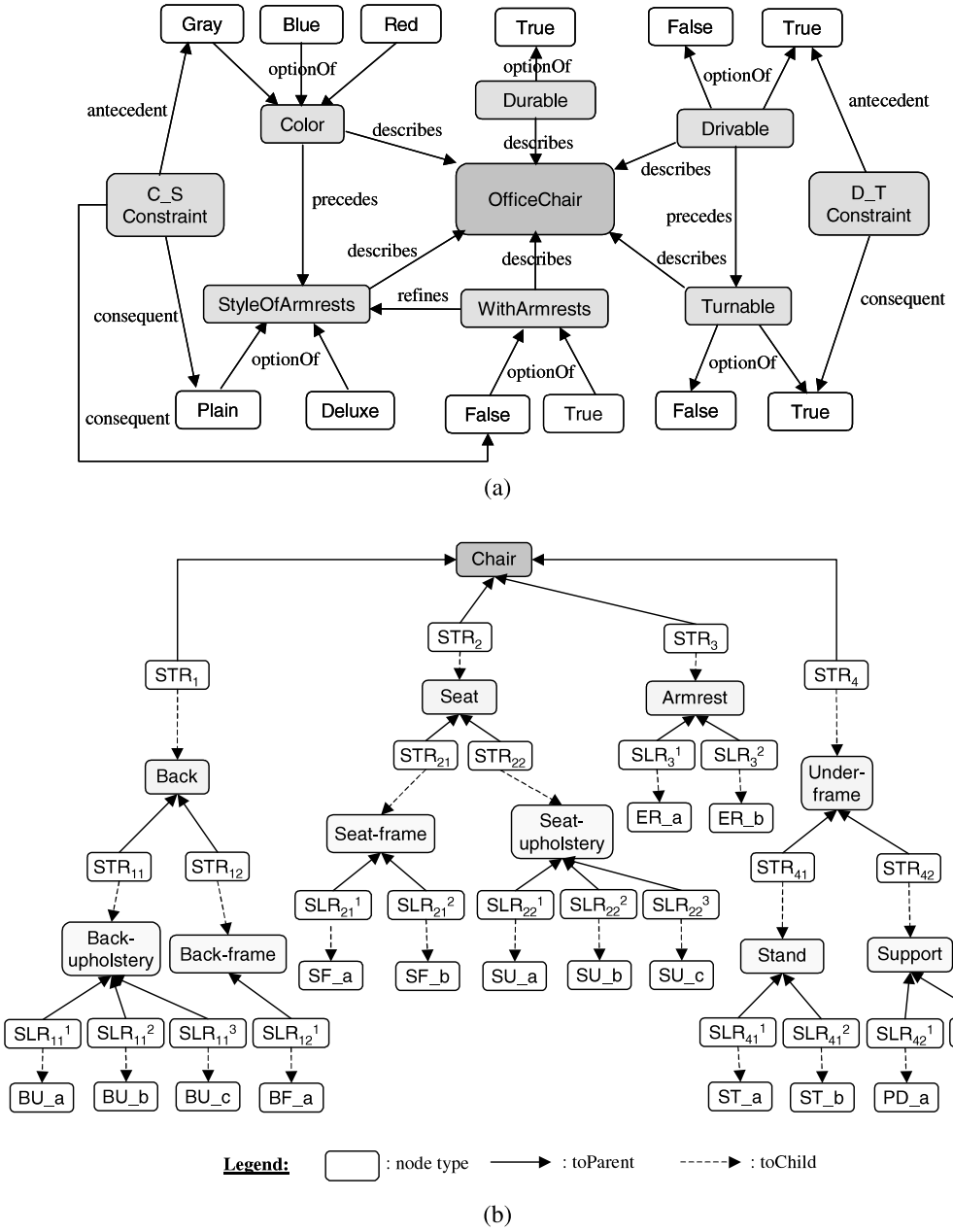


Fig. 10. (a) Sales view and (b) engineering view family graphs for the office chair product family.

attributes Turnable and Drivable of the node type Under-Frame are determined by the attributes of its parent node Chair. Similarly, a derived attribute Turnable of the node type Stand is determined by the attribute of its parent node UNDERFRAME.

Node types and their associated edges comprise the generic model of a product family, which can be represented as family graphs. The sales view family graph consists of family-specific node types for features, options, selection constraints, and product specification. The engineering view family graph consists of family-specific node types for modules, primitive variants, structural and selective relationships, and the end product. Figure 8 gives an example of a product family specification in the textual form, as well as the family graphs of this product family. In the illustration the initial values of attributes Option_{13} and Option_{22} are set to Selected, which implies that Option_{13} and Option_{22} are the default values of F_1 and F_2, respectively.

Table 2. *Modules and variety parameters in the office chair product family*

	Module	Variety Parameter
Compound module	Chair	Color, Turnable, Drivable, Durable, WithArmrests, StyleOfArmrests
	Back	Color = Chair.Color
	Seat	Color = Chair.Color, SupportArmrests = Chair.WithArmrests
	Under-frame	Durable = Yes, Turnable = Chair.Turnable, Drivable = Chair.Drivable
Primitive modules	Armrests	StyleOfArmrests = Chair.StyleOfArmrests
	Back frame	Nil
	Back-upholstery	Color = Back.Color
	Seat-frame	Nil
	Seat-upholstery	Color = Seat.Color
	Stand	Turnable = Underframe.Turnable
	Support	Drivable = Underframe.Drivable

6. CASE STUDY

Using office chairs to illustrate product families was first introduced by McKay et al. (1996). This study employs office chairs as running examples, because they are illustratively simple, yet representative. The original case is modified somewhat to highlight the characteristics of this research. Because the meta models discussed in Section 4 are universal and independent of any specific product family, this section only focuses on the generic models of office chair product families and how variant products are derived based on generic models and defined graph operations.

6.1. Family graphs

Suppose six features (F_0, F_1, F_2, F_3, F_4, and F_5) can be used for customers to specify an office chair, including color, whether a chair is drivable or turnable, if it has armrests, and the style of armrests. Two selection constraints are introduced, which are listed in Table 1 together with features and their selectable options. Figure 9(a) gives the sales view specification of node types in the generic model. Figure 10(a) shows the corresponding family graph.

From an engineering viewpoint, an office chair is an assembly of a back frame, back upholstery (in different colors), a seat frame (supporting armrests or not), seat upholstery (in different colors), armrests (in different styles), a stand (turnable or not), and wheels or pads (drivable or not). The variety parameters and associated modules are listed in Table 2. All available primitive variants and associated inclusion conditions are listed in Table 3.

Figure 9(b) gives the textual specification of node types for the engineering view generic model. In the illustration, composes is a derived path from a child node to its parent node via a structure relationship. Figure 10(b) shows the corresponding engineering view family graph. In the illustration we use subscripts to differentiate various structural and selective relationships.

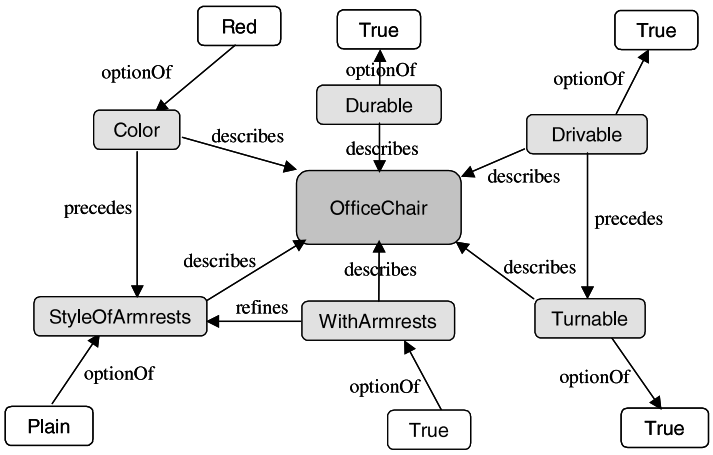
6.2. Product variants

After family graphs are constructed, they can serve as the starting graphs of graph grammars defined for a specific product family. While family graphs indicate all node and edge labels and node attributes, other elements of graph grammars such as productions and control diagrams are defined in the meta model and thus can be adapted to the generic model. Therefore, product variant derivation becomes a series of graph transformations. All variants that can be derived through graph rewriting embody the product family.

Suppose a customer presents his choices: {Color = Red, Turnable = Yes, Drivable = Yes, WithArmrests = Yes, StyleOfArmrests = Plain}. According to this input, the PFD system can generate a particular design for this customer. Figure 11(a) shows the sales view variant graph resulting from the sales view family graph in Figure 10(a) with all unselected options and features removed. The transfor-

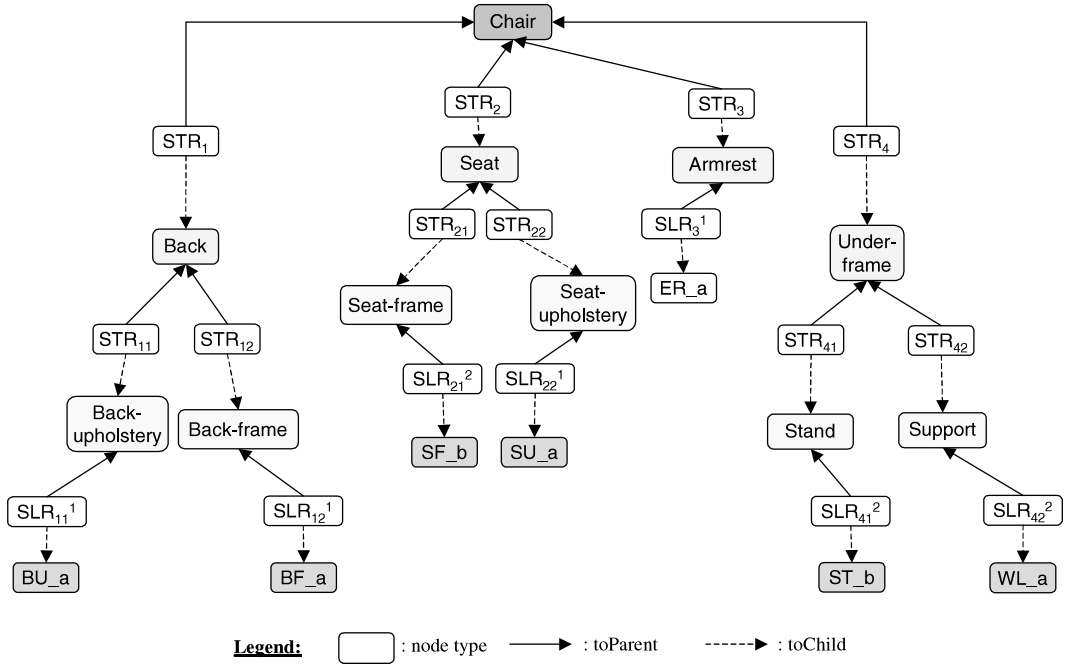
Table 3. *Primitive variants and inclusion conditions associated with the office chair product family*

Primitive Variant	Inclusion Condition
BU_a	BackUphostery.Color = Red
BU_b	BackUphostery.Color = Blue
BU_C	BackUphostery.Color = Grey
SF_a	SeatFrame.SupportArmrests = No
SF_b	SeatFrame.SupportArmrests = Yes
WL_a	Support.Drivable = No
PD_a	Support.Drivable = Yes
SU_a	SeatUphostery.Color = Red
SU_b	SeatUphostery.Color = Blue
SU_c	SeatUphostery.Color = Grey
ER_a	Armrests.StyleOfArmrests = Plain
ER_b	Armrests.StyleOfArmrests = Deluxe
ST_a	Stand.Turnable = No
ST_b	Stand.Turnable = Yes



`Input:` {Color = Red; Turnable = Yes; Drivable =Yes; WithArmrests = Yes; StyleOfArmrests = Plain}

(a)



`Specification:` {Color = Red; Turnable = Yes; Drivable =Yes; WithArmrests = Yes; StyleOfArmrests = Plain}

(b)

Fig. 11. (a) Sales view and (b) engineering view variant graphs for a custom office chair.

mation of Figure 10(a) to Figure 11(a) demonstrates the graph rewriting process from a starting graph to a variant graph.

The engineering view variant graph is derived by transforming the engineering view family graph in Figure 10(b) according to the control structure defined in Figure 6(b). The resulting graph is shown in Figure 11(b).

Each specific design represented as a variant graph like the one in Figure 11(b) can be transformed to a traditional BOM structure. For this purpose, we introduce two additional productions to change the STRUCTURAL_RELATIONSHIP and SELECTIVE_RELATIONSHIP nodes along with the relevant edges in a variant graph to the edges in a BOM structure representing the goes-into relationships among modules. As illustrated in Figure 12(a), a production, BuildUpParentChildRelation, is defined to change a subgraph consisting of the

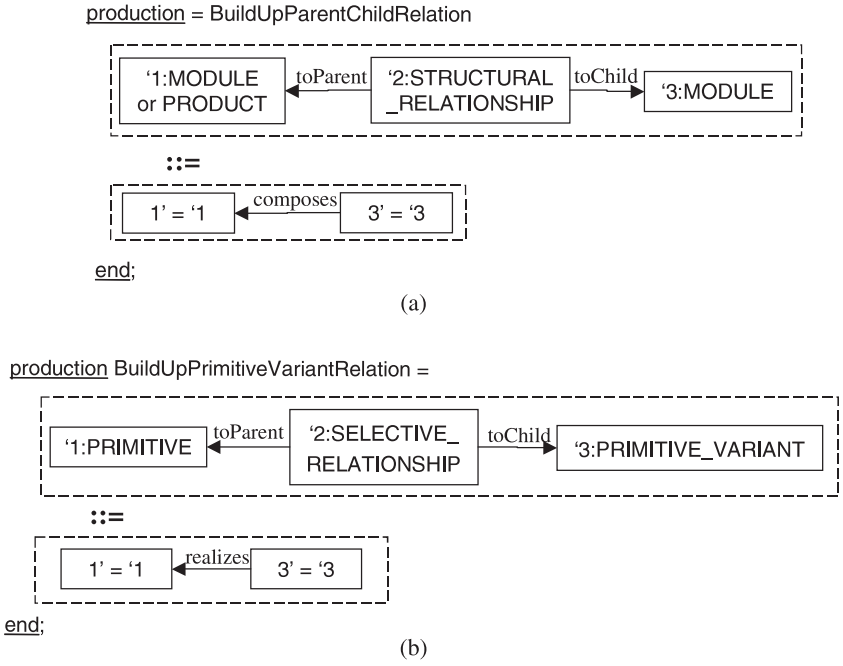


Fig. 12. Productions for transforming variant graphs to BOM structures: (a) changing a STRUCTURAL_RELATIONSHIP node to a composes edge and (b) changing a STRUCTURAL_RELATIONSHIP node to a realizes edge.

STRUCTURAL_RELATIONSHIP node and its relevant edges to a composes edge. As shown in Figure 12(b), the BuildUpPrimitiveVariantRelation production is used to change a subgraph consisting of the SELECTIVE_RELATIONSHIP node and its relevant edges to a realize edge. Applying these productions to the variant graph in Figure 11(b), a BOM-like graph can be generated as shown in Figure 13.

7. SUMMARY

Based on the understanding of product family architecture (Du et al., 2001) and the rationale of graph-based modeling (Du, 2000), this paper develops a graph rewriting system to support PFD. The modeling of PFD is approached from multiple perspectives. In the sales view, PFD is represented as a sales view family graph. This graph consists of a set of nodes representing features, options, and selection constraints, together with a set of edges indicating their interrelationships. In the engineering view PFD is modeled as an instantiation process of a GPS, which is represented as an engineering view family graph. This graph consists of a set of nodes representing the end product, modules, primitive variants, structural relationships, and selective relationships, combined with a set of edges denoting the interrelationships among them. These family graphs act as the starting graphs for a series of graph operations through which variant graphs can be derived by executing predefined rewriting rules according to appropriate control structures. Each variant graph corresponds to a customer-specific product variant.

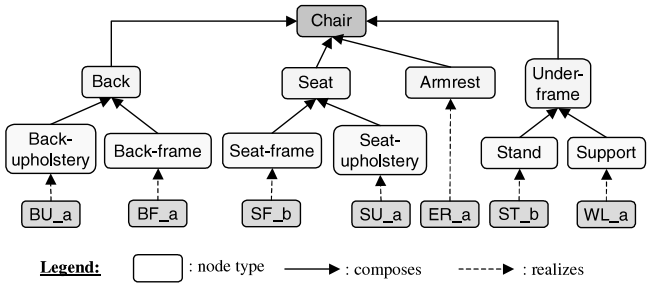


Fig. 13. The BOM structure derived by graph rewriting.

The PFD system is specified using PROGRES. As a stratified typed language, PROGRES distinguishes between node classes, node types, and node instances (Heimann et al., 1995). This allows us to define a meta model for family graphs, which factors out those entities common to all product families; define a generic model describing all specific entities relevant to particular families; and obtain a data structure of product families describing every product variant for individual customer orders. PROGRES excels in modeling major aspects of PFD systems. Attribute dependency is used to model parameter propagation in the GPS. The hierarchical graph schema supports multiple inheritances of graph entities. Parametric rewriting rules support controlled use of formal (node) type parameters within generic subgraph tests and graph rewriting rules.

The development of the prototype system for office chairs demonstrates that the graph-based PFD system can provide an interactive environment for customers to make choices among product offerings. It also facilitates design automa-

tion of product families and enhances interactions and negotiations among sales, design, and manufacturing.

ACKNOWLEDGMENTS

This research was partially supported by the Hong Kong Government Research Grant Council (HKUST 797/96E and HKUST 6220/99E). The authors would like to express their sincere thanks to Professor Martin Helander, Professor Derick Wood, Dr. Benjamin Yen, and Professor Farrokh Mistree for their valuable advice. Dr. W. K. Lo and Mr. Xuanzhong Liu at Artesyn Technologies Asia-Pacific Ltd are also acknowledged for their continuous support.

REFERENCES

Baldwin, R.A., & Chung, M.J. (1995). Managing engineering data for complex products. *Research in Engineering Design 7(4)*, 215–231.

Brown, D.C., & Birmingham, W. (1997). Understanding the nature of design. *IEEE Intelligent Systems & Their Applications 12(2)*, 14–16.

Cannondale. (2000). Cannondale bicycles. http://www.cannondale.com/.

Chen, W., Simpson, T.W., Allen, J.K., & Mistree, F. (1996). Use of design capability indices to satisfy a ranged set of design requirements. In *Advances in Design Automation* (Dutta, D., Ed.), 96-DETC/DAC-1090. Irvine, CA: ASME.

Customatix. (2000). Customatix shoes. http://www.customatix.com/.

Darr, T., Klein, M., & McGuinness, D.L. (1998). Special issue on configuration design. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing 12(4)*, 293–294.

Dell. (2000). Dell computers. http://www.dell.com/.

Du, X. (2000). *Architecture of Product Family for Mass Customization*. PhD Thesis. Hong Kong: Hong Kong University of Science & Technology.

Du, X, Jiao, J., & Tseng, M.M. (2001). Architecture of product family: Fundamentals and methodology. *Concurrent Engineering: Research and Application 9(4)*, 309–325.

Economist. (2000). Business: All yours. *The Economist 355(8164)*, 57–58.

Erens, F., & Verhulst, K. (1997). Architectures for product families. *Computers in Industry 33(2–3)*, 165–178.

Eyeplanet. (2000). Eyeplanet optics. http://www.eyeplanet.net/.

Fujita, K., Akagi, S., Yoneda, T., & Ishikawa, M. (1998). Simultaneous optimization of product family sharing system structure and configuration. *Proc. 1998 ASME Design Engineering Technical Conferences*, DETC98/DFM-5722, ASME, Atlanta, GA.

Haworth. (2000). Haworth furniture. http://www.haworth-furn.com/.

Hegge, H.M.H., & Wortmann, J.C. (1991). Generic bill-of-material: A new product model. *International Journal of Production Economics 23(1–3)*, 117–128.

Heimann, P., Joeris, G., Krapp, C., & Westfechtel, B. (1995). A programmed graph rewriting system for software process management. *Electronic Notes in Theoretical Computer Science*. http://www.elsevier.nl/locate/entcs/volume2.html.

Huffman, C., & Kahn, B.E. (1998). *Variety for Sale: Mass Customization or Mass Confusion?* Cambridge, MA: Marketing Science Institute.

Idtown. (2000). Customized watches. http://www.idtown.com/.

Jiao, J., & Tseng, M.M. (1999). An information modeling framework for product families to support mass customization production. *CIRP Annals 48(1)*, 93–98.

Jiao, J., Tseng, M.M., Ma, Q., & Zou, Y. (2000). Generic bill of materials and operations for high-variety production management. *Concurrent Engineering: Research and Application 8(4)*, 297–322.

Kota, S., & Sethuraman, K. (1998). Managing variety in product families through design for commonality. *Proc. 1998 ASME Design Engineering Technical Conferences*, DETC98/DTM-5651, Atlanta, GA.

McKay, A., Erens, F., & Bloor, M.S. (1996). Relating product definition and product variety. *Research in Engineering Design 2(2)*, 63–80.

Meyer, M.H., & Lehnerd, A.P. (1997). *The Power of Product Platform—Building Value and Cost Leadership*. New York: Free Press.

Meyer, M.H., & Utterback, J.M. (1993). The product family and the dynamics of core capability. *Sloan Management Review 34(3)*, 29–47.

Mittal, S., & Frayman, F. (1989). Towards a generic model of configuration tasks. *Proc. 11th International Joint Conference on Artificial Intelligence*, pp. 1395–1401. San Francisco: Morgan Kaufmann.

Pine, B.J. (1993). *Mass Customization: The New Frontier in Business Competition*. Boston: Harvard Business School Press.

Robertson, D., & Ulrich, K. (1998). Planning for product platforms. *Sloan Management Review 39(4)*, 19–31.

Rothwell, R., & Gardiner, P. (1990). Robustness and product design families. In *Design Management: A Handbook of Issues and Methods* (Oakley, M., Ed.), pp. 279–292. Cambridge, MA: Basil Blackwell Inc.

Sabin, D., & Weigel, R. (1998). Product configuration frameworks—A survey. *IEEE Intelligent Systems & Their Applications 13(4)*, 42–49.

Sanderson, S., & Uzumeri, M. (1995). Managing product families: The case of the Sony Walkman. *Research Policy 24(5)*, 761–782.

Sawhney, M.S. (1998). Leveraged high-variety strategies: From portfolio thinking to platform thinking. *Journal of the Academy of Marketing Science 26(1)*, 54–61.

Schonfeld, E. (1998). The customized, digitized, have-it-your way economy. *Fortune 138(6)*, 115–124.

Schürr, A. (1990). PROGRES: A VHL-language based on graph grammars. *Proc. 4th Int. Workshop on Graph-Grammars and Their Application to Computer Science*, pp. 641–659. LNCS 532. Berlin: Springer Verlag.

Schürr, A. (1994). Rapid programming with graph rewrite rules. *USENIX Symposium on Very High Level Languages*, pp. 83–100. Berkeley, CA: USENIX Association.

Schürr, A., Winter, A., & Zündorf, A. (1995). Graph grammar engineering with PROGRES. *Proc. 5th European Software Engineering Conference* (Schäfer, W., & Botella, P., Eds.), pp. 219–234. Barcelona, Spain, LNCS 989. Berlin: Springer–Verlag.

Schürr, A., Winter, A., & Zündorf, A. (1998). PROGRES: Language and environment. In *Handbook on Graph Grammars: Applications* (Rozenberg, G., Ed.), Vol. 2, pp. 1–61. Singapore: World Scientific.

Simpson, T.W., Chen, W., Allen, J.K., & Mistree, F. (1996). Conceptual design of a family of products through the use of the robust concept exploration method. *Proc. 6th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, Vol. 2, pp. 1535–1545. AIAA-96-4161-CP. Bellevue, WA: AIAA.

Simpson, T.W., Maier, J.R.A., & Mistree, F. (1999). A product platform concept exploration method for product family design. *Proc. 1999 ASME Design Engineering Technical Conferences*, DETC99/DTM-8761, Las Vegas, NV.

Spira, J.S. (1993). Mass customizing through training at Lutron Electronics. *Planning Review 22(4)*, 23–24.

ThinkCustom. (2000). *Think custom conference*. February 22–23, New York. http://www.thinkcustom.org/presentation.htm.

Ulrich, K. (1995). The role of product architecture in the manufacturing firm. *Research Policy 24(3)*, 419–440.

Ulrich, K., & Eppinger, S.D. (1995). *Product Design and Development*. New York: McGraw–Hill.

van Houten, F.J.A.M., Tomiyama, T., & Salomons, O.W. (1998). Product modeling for model-based maintenance. *CIRP Annals 47(1)*, 123–128.

Volvo. (2000). Design your own Volvo. http://www.volvocars.com/ and http://new.volvocars.com/new/design/index.html.

Whitney, D.E. (1993). Nippondenso Co. Ltd: A case study of strategic product design. *Research in Engineering Design 5(1)*, 1–20.

Wortmann, J.C., & Erens, F.J. (1995). Control of variety by generic product modeling. *Proc. First World Congress on Intelligent Manufacturing Processes and Systems*, Vol. 2, pp. 1327–1342. University of Puerto Rico, Puerto Rico, USA.

Xuehong Du is Sales Manager at Artesyn Technologies Asia-Pacific Ltd, Hong Kong. She received her PhD degree from the Department of Industrial Engineering & Engineering Management at the Hong Kong University of Science & Technology. She holds BEng and MEng degrees in Electrical Engineering from Xi'an Jiaotong University, Peoples Republic of China. She previously worked as a lecturer at

the School of Management in Xi'an Jiaotong University, China. Her research interests include design theory and methodology, mass customization, project management, and global manufacturing.

Jianxin Jiao is Assistant Professor of Systems and Engineering Management in the School of Mechanical and Production Engineering at Nanyang Technological University, Singapore. He received a PhD from the Department of Industrial Engineering and Engineering Management at the Hong Kong University of Science & Technology. He holds a BEng degree in Mechanical Engineering from Tianjin Institute of Light Industry in China, and a MEng degree in Mechanical Engineering from Tianjin University in China. He has worked as a lecturer in the Department of Industrial Engineering at Tianjin University in China. His research interests include design theory and methodology, manufacturing systems engineering, mass customization, and global manufacturing.

Mitchell M. Tseng is Professor and Head of the Department of Industrial Engineering and Engineering Management, Hong Kong University of Science & Technology. He holds a BS degree in Nuclear Engineering from the National Tsing Hua University in Taiwan and MS and PhD degrees in Industrial Engineering from Purdue University. He joined the Hong Kong University of Science and Technology as the Founding Department Head of Industrial Engineering in 1993 after holding executive positions at Xerox and Digital Equipment Corporation for almost two decades. He previously held faculty positions at the University of Illinois at Champaign–Urbana and the Massachusetts Institute of Technology. His research interests include global manufacturing systems, mass customization, information systems applications, and business process design.