4th WDK Workshop on Product Structuring
October 22-23, 1998
Delft University of Technology
Delft, The Netherlands

MODELING CONFIGURABLE PRODUCT FAMILIES

Juha Tiihonen, Timo Lehtonen, Timo Soininen, Antti Pulkkinen, Reijo Sulonen, and Asko Riitahuhta

Product configuration, product variation, product modeling, conceptual analysis, case study

Abstract

This paper presents a method for managing and modeling a product family as a configurable product. The method enables efficient management of a large number of product variants. The modeling is based on a recently proposed conceptualization of configuration domain, which is a synthesis and extension of the main approaches to configuration. The concepts of the conceptualization are components, attributes, resources, ports, contexts, functions and constraints. In addition to discussing the concepts, we give guidelines on using them. The conceptualization was evaluated through a case study of modeling rock drilling equipment.

The conceptualization matched the modeling needs in the case product. Some improvements to the conceptualization are proposed on the basis of the case study. The guidelines should also be extended and refined. Information technology support for modeling was found to be necessary for full-scale use of the conceptualization.

The conceptualization is primarily intended for representing configuration models. However, we present some possibilities for utilizing the conceptualization in the product development process.

1. Introduction

In many industries competitiveness requires efficient design and delivery of large numbers of product variants. One-of-a-kind products or a large number of fixed products often lead to excessive amounts of design and customer specific engineering, or problems with the management of a large number of product variants.

In this paper we present a method for managing large product families as a configurable product. The utilization of configurable products requires a systematic sales-delivery process and modeling the product family as a configurable product. Instead of explicitly defining a set of product variants in a product family, a configurable product has a configuration model that contains all the information on the possibilities of adapting the product to customer needs. The *configuration model* defines a set of pre-designed components, rules on how these can be combined into valid product variants and rules on how to achieve the desired functions for a

customer. However, the product needs to be designed and systemized with configurability in mind in order to make the configuration models manageable. Configurable products have become popular with companies as a means to satisfy a wide range of customer requirements, reduce costs and decrease the lead-times in the sales-delivery process [1].

In this paper we first briefly discuss the main characteristics of configurable products. We then describe a recently proposed conceptualization [2] of configuration models, configurations and requirements on a configuration. Guidelines on how the concepts are used in modeling are given. Our conceptualization unifies many of the different conceptualizations that have been proposed in AI-oriented research on configuration (see e.g.[3]). These approaches are further generalized on the basis of our experiences with real world products.

We demonstrate the conceptualization and evaluate it using a case study of modeling a product in the field of mobile machinery. The conceptualization matched the modeling needs in the case product. Some improvements and extensions to the conceptualization are proposed on the basis of the case study. The guidelines should be extended and refined. Information technology support for modeling was found to be necessary for full-scale use of the conceptualization.

The conceptualization is primarily intended for representing the product knowledge needed in the sales-delivery process. However, we argue that the conceptualization can also be used in the product development process. The main benefits arise from improved communication between product development team and other functions of the company. It is also possible to use the conceptualization in a Design For Configuration (DFC) tool.

2. Configurable products and configuration process

According to our definition a *configurable product* has the following basic properties:

- Each delivered product individual is tailored to the individual needs of an individual customer.
- The product has been pre-designed to meet a given range of different customer requirements.
- Each product individual is specified as a combination of pre-designed components or modules. Thus, there is no need to design new components as a part of the sales-delivery process.
- The product has a pre-designed general structure.
- The sales-delivery process requires only systematic variant design, not adaptive or original design in the sense of Pahl and Beitz [4].

The product development and sales-delivery processes of configurable products are separate. The product development process of a configurable product produces, among other deliverables, an explicit configuration model. However, in relatively common industrial practice the configuration models are only implicitly present as tacit knowledge in product experts' minds or in varied documents.

The configuration model is used repeatedly in the *configuration process* (Figure 1), which is a part of the sales-delivery process, to produce *configurations*, i.e. descriptions of the product individuals to be delivered. Product individuals are *configured*, i.e. adapted to meet given customer requirements, on the basis of the configuration model and the customer requirements. These adaptation activities are referred to as *configuration tasks*. The tasks are sometimes carried out in two phases in the configuration process. In *sales configuration*, the product may be partially specified in terms of functions. In *engineering configuration,* the result of sales configuration is used as an input that is refined to a technical description of the system.

3. Modeling on the basis of the conceptualization

In this section we describe our conceptualization of configuration knowledge. We define the main conceptual categories of knowledge related to product configuration. We then describe the concepts in these categories and give guidelines for their use in modeling. For more detailed definitions of the concepts we refer to [2]. We make the typographical convention that the concepts are typeset with *SMALL CAPITAL LETTERS* when they are defined and examples of concepts with arial font.

The conceptualization is intended to capture the product specification that is needed while configuring a product. Note that it does not cover knowledge related to geometry, pricing, optimality of configurations or knowledge needed during product development process. Neither does it cover the construction and control knowledge on how to accomplish the configuration task. There is little or no support for modeling of dynamic product behavior. Trying to model these aspects with the conceptualization would probably lead to major difficulties.

3.1 Configuration knowledge categories

We distinguish between three categories of configuration knowledge: configuration solution knowledge, configuration model knowledge, and requirements knowledge. *Configuration solution knowledge* specifies a (possibly partial) configuration. A configuration does not only represent the final outcome of the configuration process but also its intermediate stages. An *incomplete* configuration describes a set of possible product individuals as it leaves some aspects about the product individual open. A *specific configuration* represents a single variant.

Configuration model knowledge is represented as configuration models. Thus, configuration model knowledge specifies the set of *correct*[1] configurations of a product with respect to the configuration model and requirements.

Requirements knowledge specifies the requirements on the configuration to be constructed. Requirements knowledge can in our view be specified with the same concepts as configuration model knowledge and configuration solution knowledge, although it plays a different role in problem solving. For this reason, we explicitly discuss only configuration model knowledge and configuration solution knowledge when describing the concepts. Note that in a broad sense the set of different requirements that the product can satisfy are already set when developing the

[1] We omit here the precise definition of a 'correct' configuration. For a more detailed analysis of the correctness of a configuration with respect to a subset of this conceptualization, we refer to [5].

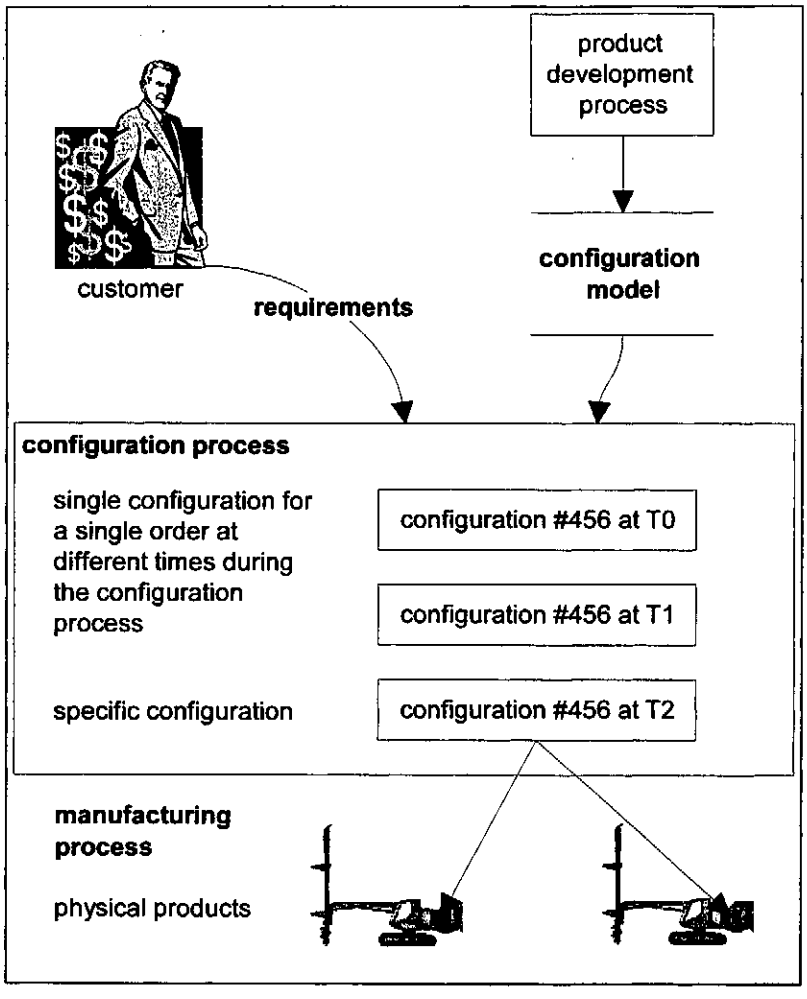

Figure 1. Configuration process

product and its configuration model. The requirements knowledge referred to here is a systemized representation of the requirements that can be set on an individual configuration. A configurer must translate the actual requirements of a customer to the elements available in a configuration model.

3.2 Modeling guidelines

A configuration model is based on an analysis of the product to be modeled. Therefore the modeler should have a good understanding of the product. The product should be modeled by product experts in the product development process. The configuration model is an abstraction of the real world product family that is specifically meant for configuration purposes. For example, it may suffice to model the different types of motors of drill rigs as simple undivisible components although they are very complex assemblies. When a drill rig is configured, it is enough to decide the type of the motor.

We consider object-oriented analysis and modeling (e.g. [6]) a good method for configuration knowledge modeling with the conceptualization. In the first phase the primary concern is to

recognize the relevant entities. These are subtypes of the concepts in the conceptualization. After that, and to some extent in parallel with the previous stage, relationships between the entities and their properties are identified. Again, the conceptualization provides the relevant properties and relations. In parallel with the previous steps, classification hierarchies of component, resource, port and function types are constructed to capture common characteristics. Constraints are identified and modeled.

For all the concepts, the rule is: use as few types and definitions in a model as possible while making sure that the model 1) contains all the necessary knowledge, and 2) is understandable and manageable. The second requirement is subjective and requires both practice and good understanding of the product to be modeled and of its usage. It is also evident that different needs lead to different models of the same product.

3.3 Classification hierarchy

3.3.1 Concepts

We introduce the concepts of a *TYPE* and an *INDIVIDUAL* to clearly distinguish between the entities that occur in configuration model knowledge and configuration solution knowledge. Configuration knowledge is commonly discussed using terms that do not distinguish between configuration model knowledge and configuration solution knowledge. For example, the sentence "Car has an engine as a part" can be interpreted in two ways. As configuration model knowledge the sentence can be understood as saying that every car individual must have an engine individual as a part. As configuration solution knowledge it states that a particular configuration includes a particular car individual that has a particular engine individual as a part.

In the conceptualization a configuration can contain individuals of four different types in the configuration model knowledge: component, port, resource and function. These are organized in a classification hierarchy in the usual manner [6]. A type has a set of *property definitions* such as attribute, part and port definitions. A type *inherits* the properties of its *supertypes* in the classification hierarchy. In other words properties from each supertype of a type are "added" to those of the type itself. For example, type Motor may define properties common to all motors of Ford Mondeos™. Types Zetec_2.0_16V™ and Endura_TD_1.8™ would have the common properties defined by Motor. We say that Zetec_2.0_16V and Endura_TD_1.8 are *direct subtypes* of Motor. Motor is a *direct supertype* of Zetec_2.0_16V and Endura_TD_1.8. All direct subtypes of a type and their subtypes, and so on are collectively called *subtypes* of a type. Supertypes are defined analogously. Multiple inheritance, i.e. having several direct supertypes, may cause complications, which are ignored here.

A subtype may *refine* the property definitions it has inherited. Refinement of property definitions is semantically based on the notion that the set of potential valid individuals directly of the subtype is smaller than the set of valid individuals directly of the supertype. For example, type Mondeo might define that applicable motors are the whole line of motors available. Type Mondeo_CL might limit compatible motors to smaller ones and Mondeo_GLX to larger ones.

A type is either abstract or concrete. An individual directly of a *concrete type* is accurate enough to be used in an unambiguous configuration. An individual directly of an *abstract type* provides only partial information on the real world entity it represents. A specific configuration contains

only individuals directly of concrete types to unambiguously specify a product individual. Note that a concrete type may allow variation in the properties of the individual, for example different individuals of a concrete type can have different attribute values.

3.3.2 Modeling guidelines

Classification hierarchies should be used when there is a set of types that have common properties or that need to be referred to collectively in some definitions. A model can be made more compact and maintainable by collecting the common properties to supertypes.

There are always several criteria for classification and one has to choose the most useful ones. Classification in product configuration models should be constructed from the point of view of configuration to facilitate easy construction and maintenance of configuration models. For example, classification hierarchies of part library standards such as ISO13584 and classification constructed for the purpose of a company's general PDM may not be appropriate for configuration models.

One should consider carefully which properties belong to a given type and which ones to its subtypes or supertypes. Common properties should be presented as high in the classification hierarchy as possible. Classification requires a good understanding of the domain being modeled.

Abstract types naturally emerge when the knowledge common to a set of types is gathered in a supertype. Quite often intermediate (abstract) types can be used to represent properties common to a set of types, but intermediate types should not be created without proper justification. For example, we do not recommend using intermediate types like "motors usable in Ranger_500", because this is best accomplished using part definitions (see Section 3.5).

3.4 Attributes

3.4.1 Concepts

Component, port, resource and function types can define attributes. Attributes represent the characteristics of an individual of the type. Some attributes have fixed values and others can be given a value. An *ATTRIBUTE DEFINITION* consists of an *ATTRIBUTE NAME,* an *ATTRIBUTE VALUE TYPE* and a *NECESSITY DEFINITION.* In a correct configuration an individual of the type having an attribute definition has an *ATTRIBUTE* according to the attribute definition and either exactly one (*necessary* attribute) or at most one (*optional* attribute) *ATTRIBUTE VALUE* of the attribute value type. One particularly important attribute type is *PHYSICAL QUANTITY* such as length or mass.

3.4.2 Modeling guidelines

Typical attributes include physical dimensions of parametric components, surface material, color, resistance, and capacity. For example, an engine could have weight and volume as attributes.

Attributes can be used to parameterise components with respect to some properties. In some cases one can create either one or a few parametric types or a relatively large number of non-parametric types. In extreme cases one parametric component type can represent millions of

non-parametric types. A large set of similar component types in a model may indicate that the set could be represented as a parametric component type.

3.5 Components and structure

In this section we first motivate the need for advanced product structure modeling. Then we define the central concepts of component types, component individuals and their compositional structure.

One of the most fundamental aspects of any technical system is its hierarchical decomposition. Fixed part lists or bills of material (BOM) with no support for variation are commonly used to describe the structure of both fixed and one-of-a-kind products. This approach becomes unmanageable when the number of different structures becomes very large.

Configurable products make the management of very large numbers, even millions or more, of variants efficient. This is due to a configuration model defining its product variants in a combinatorial manner. For example, a configuration model representing 4 options that one may either include in the product individual or not and 5 choices with 4 alternatives for each choice in effect represents $2^4 \times 4^5 = 16384$ variants. Changing one of the alternatives for one choice would affect 4096 fixed structures if the product family were modeled as fixed products. In a configuration model only one change could be required.

3.5.1 Concepts

A *COMPONENT TYPE* represents a distinguishable entity in a product that is meaningful for product configuration. A configuration is composed of *COMPONENT INDIVIDUALS* of the component types in a configuration model. Component types are either *dependent* or *independent*. Only individuals of independent types can serve as roots of the product structure.

The conceptualization directly supports generalized product structures with varying number of mandatory, alternative and optional parts. A component type can define roles for its parts. These roles are filled by component individuals in configurations. As an example, component type Lamp has part roles Lampshade and a Stand. Each part role defines which component types are viable alternatives for that part role. For example, the lampshade may be a blue lampshade LsBlue or a black lampshade LsBlack. Component individuals are assigned to fill the part role in a configuration. For example LsBlack456 fills the role Lampshade of Lamp123. No other component individuals than those of the types defined to be possible lampshades may fill the lampshade role. A component individual can only have the parts defined by roles in its type. For example, one cannot have a configuration where an umbrella reflector is a part of a lamp if a corresponding part role has not been specified.

A component type specifies its part roles as a set of *PART DEFINITION*s. A part definition specifies a *PART NAME*, a *SET OF POSSIBLE PART TYPES* and a *CARDINALITY*. In addition, a part definition includes an *EXCLUSIVITY DEFINITION* and an optional *HAS PART INHERITANCE DEFINITION*. For brevity, the latter two are not discussed in this paper. The part name identifies the role. The possible part types indicate the component types whose component individuals are allowed to occur in the role. The cardinality specifies how many component individuals must

occur as parts in the role. Cardinality is expressed as a set of non-negative integer ranges. If one of these ranges includes 0, the part is said to be *optional*. Otherwise it is *mandatory*.

Examples of part definitions in types include: A Car has an Engine and a Chassis as parts and there are several alternatives for each. A Computer has a part display unit that can be one of the following: Brand A 15" Flat, Brand A 17" Super or Brand B 19" HD.

3.5.2 Modeling guidelines

A coherent whole in a product should be modeled as a component type when it

(1) is distinct in the sense that it either may or may not appear in the configuration, or

(2) is an alternative for something else, or

(3) refers to a well-defined part of the product.

Usually one should avoid modeling things that are common to all variants or fixed substructures of a component type. Common parts and fixed substructures are usually irrelevant for the configuration view of the product and can usually be managed in an MRP system.

A part definition should be included in a configuration model if it is natural to think that an individual of component type A contains an individual of component type B, and both A and B are integral wholes whose modeling is justified by the reasons given above. In general, the parts of a whole should be separable and have a distinct identity. Usually the parts of a whole are somehow different from each other. The difference can, for example, be related to the role, function, or type of the part. Most part definitions are meaningful in the configuration sense only if a choice of a component type or number of parts can be made. However, quite many products include a mandatory "base unit" that can also be modeled. It is sometimes difficult to decide whether component type A is part of component type B or vice versa. In this case, there often is a component type C that both A and B are parts of.

Note that we model product families, assemblies, sub-assemblies and atomic parts uniformly as components. Examples of mostly mechanical components include engines, gears, elevator cars, and chassis. Electrical components include computers, microprocessors, memories, ASICs, and PCBs. Non-physical component types include software systems, software modules, applets, and different insurance policies.

3.6 Ports

3.6.1 Concepts

Ports are used to model connections and compatibilities between components. The idea is that component individuals can be connected only if they have compatible interfaces. We model connection interfaces as ports. *A PORT TYPE* is a definition of a connection interface. A *PORT INDIVIDUAL* represents a "place" where in a component individual some other port individual may be connected. A port type has a *COMPATIBILITY DEFINITION* that defines a set of port types whose port individuals can be connected to the port individuals of that port type. In addition, a port type defines a set of *CONNECTION CONSTRAINTS*. Only port individuals that satisfy the

connection constraints defined by their port types may be connected to each other. A connection constraint may also specify that port individuals with given attribute values and of particular type can or must be connected. Connection constraints are a special case of constraints (see Section 0.).

A component type specifies its connection possibilities by *PORT DEFINITIONS*. A port definition specifies a *PORT NAME* for the port, a *SET OF POSSIBLE PORT TYPES*, a *CARDINALITY* and additional connection constraints. Cardinality expresses the possible number of port individuals as a set of ranges. A port definition can refine the set of compatible port types from those specified by the port type using the .

3.6.2 Modeling guidelines

The connections modeled with ports can be physical or logical. Ports are especially suitable for modeling compatibility expressed as interfaces, which is often the case for modular products.

If component individuals do not have compatible ports, they cannot be connected, but they can still exist in the same configuration. In effect, components are by default compatible but not connectible. Compatibility can be restricted to component types with compatible ports by requiring connections using constraints.

Note that a component type can offer alternative port types, which allows configuring the interface by selecting a port type. Port definitions can also be used for limiting the number of connections, because the number of port individuals can be specified and at most one connection can be made to a port individual. In some cases only the number of connection points or available space for connections is relevant without the need to model exactly the connections between the ports. This type of phenomenon may be better modeled as a resource interaction (see below).

Examples of connections that can be modeled using ports include a motor requiring a specific type of a power source. This can be modeled with appropriate in and out port types for the motor and power source and defining that only certain kinds of in port types fit certain types of out ports. Another example is a computer that can be plugged to a printer if both of them have a parallel port. The parallel port is an interface which is modeled as a port.

Quite often actual connections require connecting components like cables or pipes. It may be difficult to decide whether a connecting component needs to be modeled or can be abstracted away. This depends on whether the connecting component has alternative types, is parametric or has some other reason for being specified separately.

It may sometimes be difficult to decide whether a component type is a part of another or is connected to it via ports. One distinguishing criterion for this is that a connection between the port individuals of two component individuals is symmetric. In other words, it seems intuitively clear that if component individual A is connected to component individual B, then B is also connected to A. On the other hand, the has-part relation between component individuals is usually interpreted as antisymmetric, i.e. if component individual A has-part component individual B, then B must not have A as a part.

There is also another dependency between connections and parthood: connections are often found between components that are parts of the same whole. Therefore, if there is set of component types that are parts of the same whole in the model, one should check whether there are some connections between these component types that are relevant for configuration and should be modeled. In the other direction, if there is a set of component types in the model that can be connected to each other, it should be checked whether they should be a part of some common whole.

3.7 Resources and contexts

3.7.1 Concepts

In this section we define resource-oriented concepts, which are needed for modeling the production and use of some more or less abstract entity. The underlying idea is that some component individual(s) produce some resource and other component individual(s) use it.

Resource production and use must be either *satisfied*, in which case the quantity of resource produced must be equal to or greater than the quantity of the resource used or *balanced*, in which case the quantity of resource produced must be equal to the quantity of the resource used.

Basic resource concepts offer a simple mechanism for modeling resource production and use without regard to product structure or connections, i.e. resources are considered to be globally available. The conceptualization adds a context mechanism that makes it possible to limit resource availability to some specific set of component individuals. The motivation is that unrestricted flow of resources from producers to users may in some cases be too simplistic to model a product adequately. A resource is only available to component individuals that are in the same context as the producing component individual.

In the conceptualization, a *RESOURCE TYPE* defines the properties of the resource. A resource type has a *COMPUTATION DEFINITION* that specifies whether the resource should be satisfied or balanced and a *UNIT OF MEASURE*. In addition, the computation definition specifies how the production and use of the resource type by several component individuals are combined, possibly taking into account production by sub- and supertypes. This is done through a *TOTAL PRODUCTION FUNCTION* and a *TOTAL USE FUNCTION*. The prototypical case is that the quantities of the resource produced or used are added together to get the total quantity.

A component type specifies by *PRODUCTION DEFINITIONS* and *USE DEFINITIONS* the resource types it produces and uses. Both production and use definitions specify a *SET OF POSSIBLE RESOURCE TYPES* produced or used, a *PROPERTY DEFINITION*, a *MAGNITUDE RANGE* and a *CONTEXT DEFINITION*. The produced or used resource must be of one of the possible resource types. The property definition is a special case of constraints (see Section 0.) that specifies a restriction on the attribute values of the resource produced or used. The magnitude range specifies how much of the resource component individuals produce or use.

3.7.2 Modeling guidelines

Quite often when a component of one type requires a component of another type for the whole to operate correctly, the underlying reason is that there is a resource which is provided by the first and used by the second. Another indication of a phenomenon that can be modeled as a

resource is that there is a flow of some thing from a component type to another. However, resources are not intended for modeling the dynamic behavior of a system. Resource concepts are useful for modeling the following phenomena:

- Individuals of a component type(s) consume some **amounts** of abstract or physical resources that individuals of some other component type(s) produce. Examples include discrete physical locations, space in rack, slots, electric power and current, disk space, torque or power produced by motor.
- Individuals of a component type(s) require the **existence of an abstract service** without regard to who is producing it. Typically these are found in computer systems and similar devices. Abstract services are often shared in the sense that the existence of one producer is enough to satisfy all users. Examples include availability of a protocol or availability of an API (Application Program Interface).
- Individuals of a component type require the **existence of an individual of other component type** in the configuration. This can be modeled as abstract services by creating a resource type that is produced by the required component type(s) and consumed by the requiring component type(s). Note that this is a practical way of associating two sets of component types to each other: If there is an individual of the first set in the configuration, there must also be an individual of the other set. The number of individuals from these sets can also be easily balanced, if required.

Resource concepts are not intended for situations where there is a need to model explicitly the connections between component individuals producing and using resources. Another way of telling connections and resource interactions apart is that connections are symmetric, whereas resource interactions are asymmetric (from the producer to the user, but not vice versa).

Property definition in resource production and use definitions can be used, for example, to model different qualities of electricity. Effectively this enables permitting or rejecting the use of a resource on basis of its attributes.

Contexts are used in situations where there is an additional condition that specifies from where a resource is obtained or where it can be used. An example is a product with two subsystems that each have their own power supplies. Power supplied within one subsystem cannot be used outside of it. Although one subsystem may have surplus power, it is not available to the other subsystem. Unexpected results may occur if a configuration can have several top-level components that are intended to be roots for independent product individuals, because resource interactions between them are possible. A context should always be specified to keep resources within a product individual in such a configuration.

3.8 Functions

3.8.1 Concepts

In this section we define the function and function structure concepts for representing the functionality that the product individual provides to the customer, the user of the product or the environment in which the product individual will be situated. The concepts introduced so far are called *technical concepts* since they have risen from the technical point of view on the product.

A complex product is often configured in two stages, as discussed in Section 2. A functional specification may, for example, specify that a telecommunications switch must provide access to at least 1000 subscribers. This function is implemented by a combination of component individuals in certain relations to each other. These relations may be expressed using the technical concepts defined above.

The basic concept in the functional view is *FUNCTION TYPE*. We call a function individual a *FUNCTION*. A *PART DEFINITION* of a function type corresponds to a part definition of a component type with the exception that the possible part types must be function types. It is used to represent the hierachical breakdown of the function to subfunctions.

The relation between technical concepts and functions and their properties is expressed as *IMPLEMENTATION CONSTRAINTS*. They are a special form of general constraints (see Section 0.) Several different combinations of technical concepts may implement the same functions and one combination of technical concepts may implement several functions.

There can be constraints on how different functions and their attribute values can be combined. These *SPECIFICATION CONSTRAINTS* are a special case of the generic constraints (see Section 0.) that only refer to functional concepts. They are used similarly as other constraints to restrict the combinations of functions that a product can implement.

3.8.2 Modeling guidelines

A function type is typically an abstract characterization of the product that a customer or sales person would use to describe what the product can be used for, or what need the product satisfies. Another, a possibly more engineering-oriented source of function types is the set of effects that the product causes in its environment. Functions are usually not conveniently defined by the technical concepts. Note that only functions of function types that are defined in a configuration model can be used for describing the functional requirements on the product. All customer requirements are not necessarily described as functions: also constraints, component individuals with attribute value assignments, resources and port individuals can be used. A combination of these forms the systematized requirements knowledge related to a configuration.

Functions are distinct from resources, since the latter are meant to be used for defining the technical rules that the product must conform to. In addition, a resource type is produced by one or several component types in separation, whereas a function type may be produced by an arbitrary combination of component types and their relations.

Implementation constraints are used to define when the technical part of a configuration implements some functions. An example of an implementation constraint in the context of the case product (see Section 4) would be that the function type 'rough-terrain-operability' is implemented by the configuration having two component individuals of type 'three-edge-track' and one component individual of type 'winch'. Another example of an implementation constraint from the PC domain is that the ability to use a MS Word7® word processor requires the configuration to have enough memory, a correct operating system and a fast enough processor.

Specification constraints are used to rule out undesirable combinations of functions. Typically they originate from marketing or product policy decisions. In our view, specification constraints

should not be used to rule out technically invalid combinations. Rather, these should be ruled out with technical concepts.

3.9 Constraints

3.9.1 Concepts

Constraints are a general mechanism for specifying the interdependencies of configuration related types in the configuration model. A *constraint* is a formal rule, logical or mathematical or a mixture of these, which specifies a condition that must hold in a correct configuration. We assume the existence of a *constraint language* with enough expressive power to express the desired concepts. The only (obvious) restriction we set on a constraint language is that it must be possible to evaluate whether a constraint is satisfied, violated or its truth-value is unknown with respect to a given configuration. Special cases of constraints, has part inheritance definitions, property definitions of resource types, connection constraints, specification constraints and implementation constraints, have already been mentioned. A constraint language should provide special support for specifying these types of constraints.

Figure 2. A Ranger drilling machine

The conceptualization includes a mechanism for defining subsets of constraints, called *CONSTRAINT SETS,* that limit the allowed configurations from specific points of view. A constraint belongs to at least one constraint set. Correctness of a configuration can be checked from a given point of view by checking whether the corresponding constraint set is satisfied.

3.9.2 Modeling guidelines

Constraints are meant to be used when other constructs of the conceptualization do not capture the intended meaning adequately or conveniently. It is possible that for some slowly evolving products a simple interaction that could be captured by deeper resource or port modeling can be modeled more easily as shallow knowledge using a simple constraint.

Technical and marketing constraints are examples of constraint sets. The technical constraints limit the configurations on the basis of which combinations are technically feasible. Marketing constraints limit the combinations on the basis of product policy, i.e., which of the technically feasible combinations a company is willing to sell. Technical constraints may be further divided into, for example, technology and manufacturing constraints.

4. Example

In our example, we define a configuration model using the conceptualization. We use a heavy rock drilling machine produced by Tamrock Corp. as an example of a configurable product. The Ranger series machines look like tracked excavators (Figure 2). A Ranger consists of a body, a tracked crawler base, a boom and drilling equipment. The body is divided into power unit, cabin, fuel oil tank, and hydraulic oil tank.

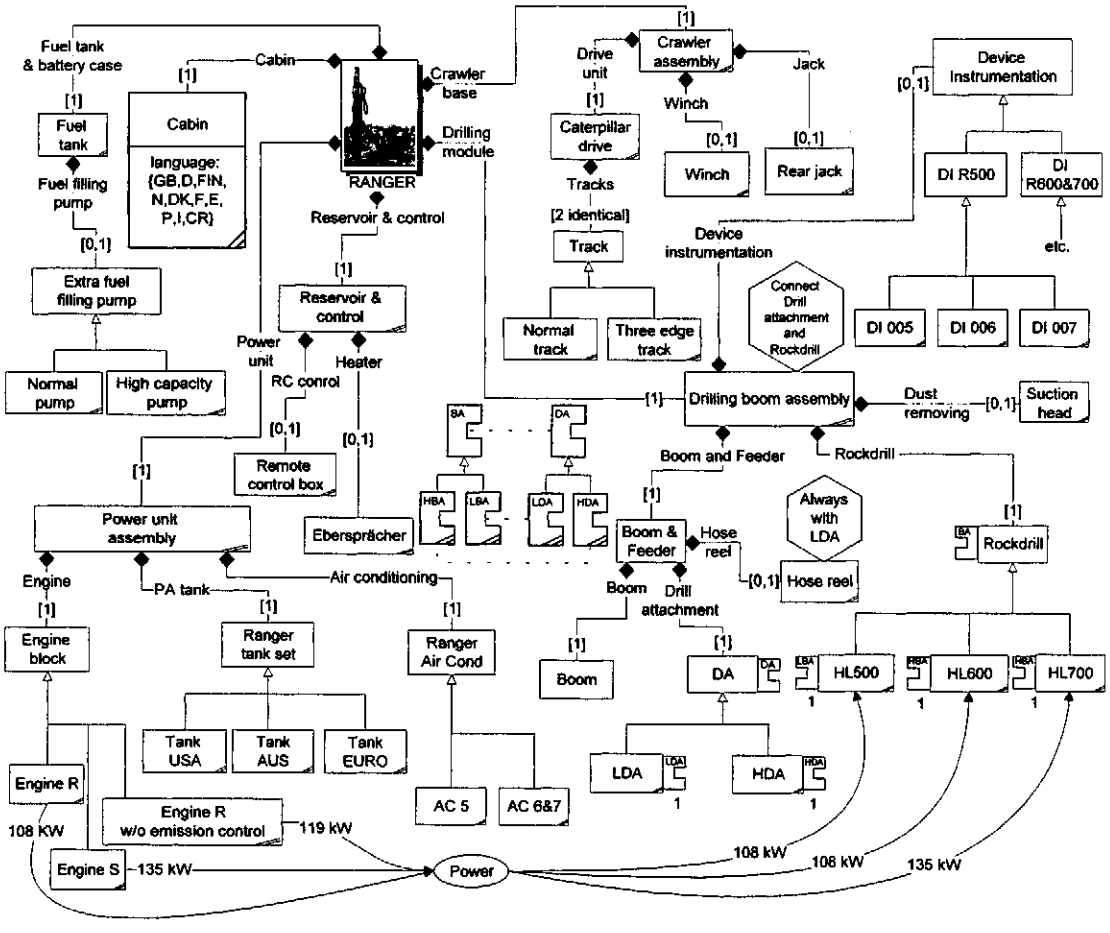

Figure 3. The Ranger configuration model

The Ranger configurable product family has three variations in the main functional property (drilling dimension) and numerous variations of minor details. Theoretically there are over 200 000 possible variants presented in the configuration model. In the real Ranger there are many more, but Tamrock Corp. regards 72 of the possible variants as substantially different product variants — this variation is also presented in the developed model.

In Figure 3 a substantial part of the configuration model of Ranger is presented. Since it is a configuration model it does not include the knowledge needed in constructing and designing the Ranger. The product is modeled mainly with part definitions. This means that components in the configuration model define the roles of their parts and what types of components are needed in these roles. Note that in the model Ranger body has been abstracted away. The notation used in the figure can be found in Appendix A.

The part definitions define the possible compositions of a Ranger individual. For the sake of simplicity only the parts that are important from the configuration point of view are presented in the model. As an example of part definition, Ranger has part role Power unit, which is filled with an instance of component type Power unit assembly. The cardinality [1] means that there is

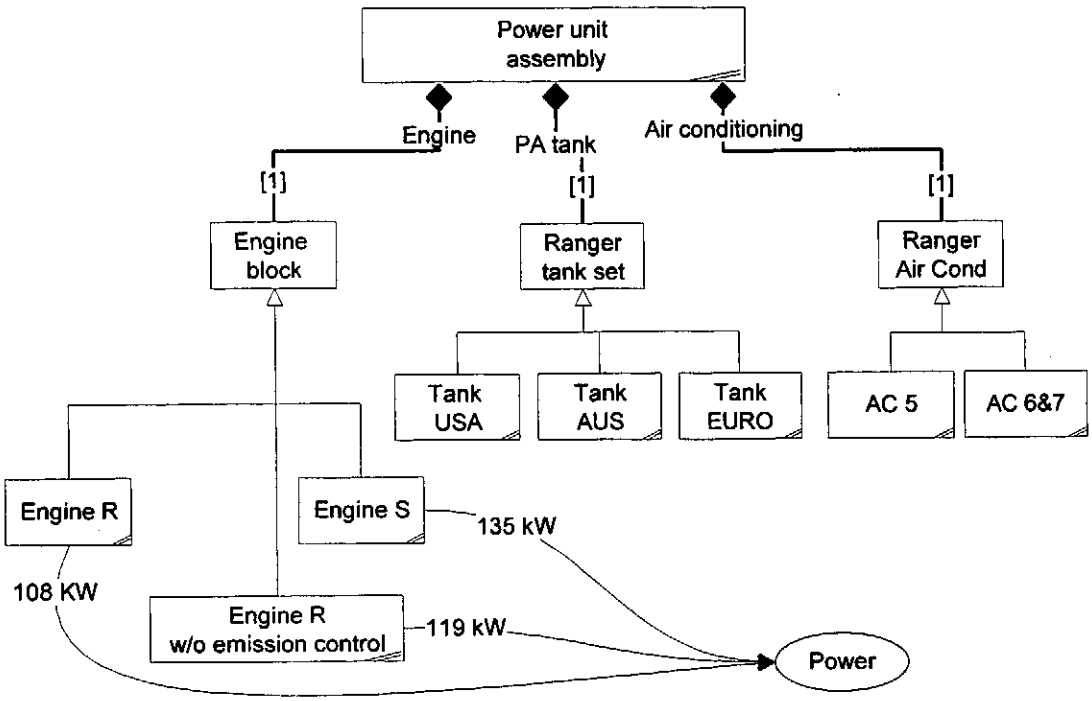

Figure 4. The Ranger Power Unit Assembly

exactly one Power unit assembly in Ranger. The Power unit assembly has three part roles Engine, PA tank and Air conditioning (see Figure 4). The role Engine could be filled with an instance of abstract component type Engine block or one of its subtypes. Here the component type Engine block is modeled as a supertype of the three concrete component types Engine R, Engine S, and Engine R w/o emission control. These types inherit the properties of engine Engine block, but differ in some aspect, e.g. they have different hydraulic pumps and differences in tuning for emission control.

Attributes are presented as shown for the Cabin (Figure 3). The language of the signs in the cabin may be different. This is represented by the Cabin type having attribute Language with several alternative values.

As mentioned earlier Ranger has three variations in its main function. The variation is caused by the capability to drill holes of different diameters. This capability is determined by the Rockdrill selection (see Figure 5). The configuration task is usually started with defining the requirements for the main function and selecting the correct type of rockdrill for that. E.g. the required hole diameter might be from 72 to 96 mm and a suitable selection the rockdrill HL600. We have ignored these aspects in the Ranger configuration model for brevity.

The type of feeder is dependent on the type of rockdrill. This dependency is modeled using port types and their compatibilities, as the rockdrill needs to be connected to mechanically the boom. For this purpose, port types *boom attachment* BA and *drill attachment* DA are introduced. BA represents the connecting interface for connecting a boom to the drill and DA represents the interface for connecting a drill to the boom. Both types are abstract types and they both have two concrete subtypes. BA has subtypes *light boom attachment* LBA and *heavy boom attach-*

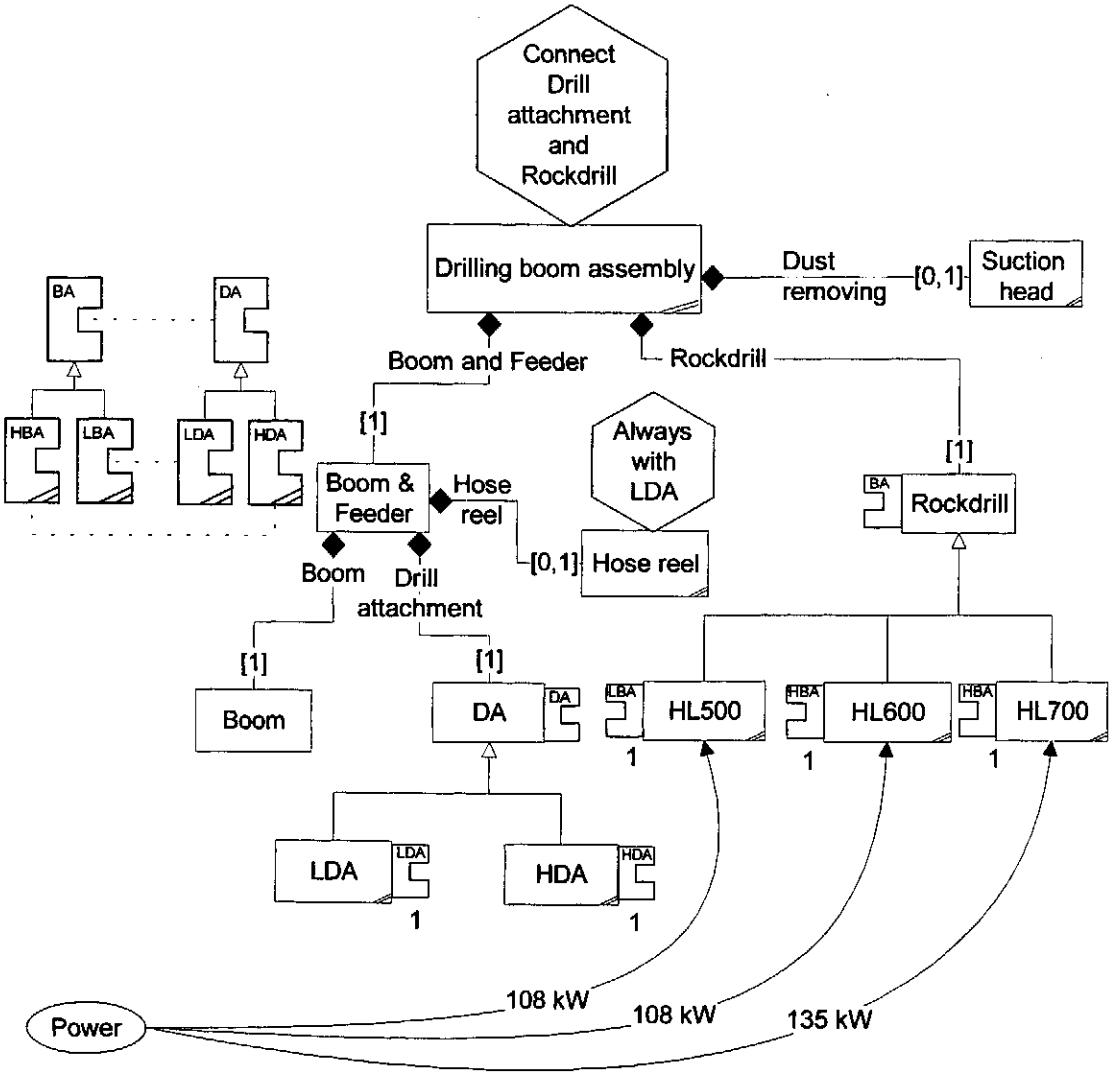

Figure 5. Drilling boom assembly

ment HBA. DA has subtypes *light drill attachment* LDA, and *heavy drill attachment* HDA (Figure 5). A port of type DA is compatible with BA, and a port of type DA is compatible with BA. Compatibility is refined in the subtypes so that LBA and LDA are compatible with each other. HBA and HDA are also mutually compatible. Effectively this means that heavy and light versions of boom and drill interfaces cannot be mixed.

Component type Rockdrill has a port of type BA and component type DA has a port of type DA. These are refined in their subtypes to LBA and HBA, and LDA and HDA, respectively. In effect, a LDA is compatible with and can be connected to only the HL500 rockdrill, whereas a HDA is compatible with and can be connected to both HL600 and HL700 rockdrills. There are no connection constraints in this example, but there is a general constraint that the Rockdrill must be connected to the Drill attachment. In addition, there is a constraint that a Hose reel component is always needed with the DA component.

The power requirement of rockdrills varies. This is modeled by using resources. The engines produce certain amounts of resource type Power which has unit kW (Figure 4). The rockdrills use this resource, represented in the bottom of Figure 5. The computation definition for Power is satisfied, which means that Power must be produced in at least the amount it is used. The contexts of Power production and use by the engines and rockdrills is in this model All, which means that any rockdrill could use Power produced by any engine within the Ranger. This has no significance in this particular model because (according the cardinalities) there is always only one engine and rockdrill in a configuration, assuming that there is only one Ranger individual. In other product families produced by Tamrock the number of (external) power sources and rockdrills varies, in which case defining contexts for the resource production and use would likely be necessary.

5. Discussion

5.1 General

Some concepts of the conceptualization overlap in the sense of formal expressiveness. For example, ports could be used to model part definitions, and a general constraint language alone could have enough expressive power for modeling configuration knowledge. In our view, the clarity of configuration models should not be compromised by minimizing the number of concepts. Higher level concepts for representing typical forms of configuration knowledge result in a more compact and understandable representation of a configuration model. We believe to have struck a good balance between minimizing the number of concepts and making configuration models understandable. This facilitates the maintenance of the knowledge by product developers or product managers. It is also possible to model a phenomenon in several ways with the conceptualization. The guidelines given in this paper are intended to help in deciding which modeling concepts to apply.

The conceptualization of product structure is quite close to *generic bills-of-material* (GBOM) (e.g. [7]). These approaches typically add to the traditional BOM concepts optional and alternative components and rules on how the alternatives and options can be combined. The main differences to the GBOM approaches are that this conceptualization

- utilizes concepts such as types, individuals and inheritance from object-oriented modeling approaches (e.g. [6]) that improve the understandability and maintainability of models,
- allows modeling connections and functions of a product,
- includes a more sophisticated means for expressing the rules on how the components can be combined, and
- represents part roles explicitly via part definitions.

Part definitions allow referring to a part of a whole without knowing the actual part to fill that role. In our view this is very important. For example, it is possible to specify that two parts must be connectable without knowing what the types of the parts will be.

We model product families, assemblies, and indivisible building blocks as components. We recognize that there is a need to distinguish between at least these different types of entities, but

our component conceptualization can be used to represent their characteristics. These kinds of concepts can easily be defined on top of our component concept.

According to our experiences, a number of configurable products have been modeled successfully with only two-level structures: wholes and their undivisible parts. We believe that most part structures in configuration models of configurable products are relatively shallow, due to the fact that the variation is accomplished through choosing the top-level components whose detailed structure is not variable. In the Ranger example, we have a three level structure: the Ranger, its main assemblies and their parts. If a product is not easy to configure, deep structures may be needed to represent minor parts that are affected by configuration.

Configuration models have been modeled as bond graphs [8]. Our port conceptualization is less expressive as we do not try to capture conversion laws, dynamic physical behavior etc. physical processes. The port mechanism for connection and compatibility modeling is relatively simple but we believe that it suffices for configuration purposes. Ports can be used to establish binary connections between component individuals, and constraints can be used to propagate values from "in-ports" to "out-ports", possibly with some transformations. However, the conceptualization has no direct support for resource flow through component individuals via port individuals, and transitive connections are not directly supported. For example, pipelines that span across several component individuals cannot be directly modeled. Possibilities offered by bond graphs could be used to add expressiveness to the conceptualization.

Interfaces between components tend to live longer than individual component types. When compatibility is modeled through stable interfaces using ports, long term management of configuration models becomes easier. Like port types, resource types may live longer than individual component types with analogous effects to maintainability. To facilitate more advanced long-term management, the conceptualization should be extended with concepts from configuration management and product data management.

5.2 Case study experiences

Classification hierarchy was found to be useful and it was used extensively in the example, particularly for component types. Multiple inheritance was not required. We defined only few attributes in the example because the company uses few parametric components and because we modeled few technical details of the component types. We believe that a full-scale configuration model for Rangers would include many more attributes to cover the technical details. Note that many legacy systems where product data is managed do not support attributes, which is reflected in the material we received and subsequently in the configuration model.

Part definitions were found to be a convenient way of representing the product. The generic structure describing Rangers was easy to construct. Similarity requirements for multiple parts would be an useful enhancement to the conceptualization. For example, Rangers have two tracks that must be identical.

Ports were used relatively little in the example. It was somewhat difficult to identify relevant ports on the basis of the available material. Ports would probably be easier to use if the interfaces of the component types in the product were originally set to high priority and thus better documented. As with ports, resource modeling was not used in Tamrock and similar

observations apply. Nevertheless, some natural resource interactions in the product were identified, of which the resource "power" is an example.

The example required relatively few general constraints. Thus, for this product our conceptualization covered the typical configuration phenomena fairly well. It should be further studied whether this result generalizes to a larger set of different kinds of products.

A computer-based tool directly supporting the conceptualization is a prerequisite for using the conceptualization successfully in real-world scale examples. Manual modeling is too tedious and offers no support for testing a model and configurations. In addition, simplicity is a challenge for the presentation and understandability of configuration models. A computer-based tool would allow a clearer graphic representation of configuration models. For example, hierarchical browsing of a configuration model, e.g. based on the product structure, separate views for classification, part definitions, ports and resources would enhance the readability of configuration models.

5.3 Ideas for improving the model and future work

There is a need to investigate several enhancements to the port mechanism. Examples include decomposition of ports to sub-ports, assignment of ports or sub-ports to implementing components and refinement of connections between ports to components. For example, the DA component type of the example in Section 4 could be modeled both as a port and a component type depending on the usage and point of view chosen. Flexible mechanisms to model components that are also ports or connectors seem to be needed.

Context concepts could be useful with other concepts than resources also, but currently they can only be used in association with resources. For example, connections via ports or applicability of constraints could be restricted to a context.

Constraint schemes and a specific constraint language for expressing common types of constraints would be useful. Examples include connection, incompatible-with and requires types of constraints. A table based constraint representation would be useful to represent tightly connected configuration decisions.

An important part in modeling some products is to ensure that connections between some parts are made. For example, it may be required to connect a fuel pump to a motor or a disk unit to a disk controller. Our conceptualization allows this to be expressed as constraints, but there is no direct support.

5.4 Relationship to product design and development

Generally, design process can be seen as an interplay between functional and physical domains [9]. It can be understood as a composition of two different activities: synthesis and analysis, which are related to the domains. The physical system is synthesized on the basis of the required functionality. Functionality can be analyzed on the basis of a description of the physical system [10]. Design process usually proceeds from general to details, which can be modeled with the conceptualization.

The conceptualization does not directly provide means for modeling physical relationships such as geometry usually used by common design tools like mechanical CAD. The conceptualization could, in principle, provide limited support for design decisions on the basis of already modeled knowledge. For example: given a partial configuration model and a component library modeled on the basis of the conceptualization, one could find suitable components by using inference from interfaces, resource behavior and functions. However, because our conceptualization is based on the assumption that all relevant knowledge is modeled, little support can be provided for decisions required in original and adaptive design.

Andreasen and Hein [12] have modeled the Integrated Product Development (IPD) process as a composition of six phases: (0) recognition of need, (1) investigation of need, (2) generation of product principle, (3) product design, (4) production preparation, and (5) execution phase. In our view, configuration models could be utilized in several phases of the IPD process. In phases (2) and (3), a preliminary configuration model consisting mostly of functions, components, part definitions and interfaces modeled as ports could be used to communicate the preliminary properties of the new product family to sales, marketing, design, and production. The creative design in the product design phase (3) is not supported by the conceptualization. In the production preparation and execution phases (4) and (5) a configuration model facilitates transferring information on the new product especially to sales and persons responsible for engineering configuration. Because the configuration model is built during the three first phases of the IPD process, some delays caused by late modeling may be eliminated.

The conceptualization could serve as a (partial) basis for a DFC (Design For Configuration) tool. The configurability and modularity of the product can be analyzed on basis of its configuration model. For example, module coupling and configuration related interfaces could be analyzed through ports and connection constraints. The number of configuration decisions can be determined and the complexity of function to technical configuration mapping can be characterized. As the usual modeling and evaluation tools in the DFX field [11], the conceptualization supports relevant modeling and enables evaluating of the design.

6. Conclusions

Managing product families consisting of a large set of product variants as configurable products requires defining a configuration model. We presented a conceptualization for configuration models and gave guidelines on using the concepts. The conceptualization was evaluated by modeling a case product and was found to cover the relevant modeling needs fairly well. However, several improvements to the conceptualization were identified. For example, more elaborate mechanisms for connection modeling would be useful. The concepts and modeling guidelines for them should be extended, refined, and further validated.

Information system support is necessary for modeling real-world-sized products. Modeling with the conceptualization sets new requirements for the designer. In addition to having a good understanding of the product, a designer should be familiar with object oriented modeling.

In our view the conceptualization can also be used in the product development process. The main benefit there would be improving communication within the product development team and to other functions of the company. The benefits also include the use of the conceptualization

to document the knowledge on the product functions, structure and the related design constraints during the product development.

References

[1] Tiihonen, J., Soininen, T., Männistö, T. and Sulonen, R. Configurable products - Lessons learned from the Finnish Industry. In *Proceedings of 2nd International Conference on Engineering Design and Automation*. Integrated Technology Systems,Inc., 1998.

[2] Soininen, T., Tiihonen, J., Männistö, T. and Sulonen, R. Towards a General Ontology of Configuration. In *AI EDAM (Artificial Intelligence for Engineering Design, Analysis and Manufacturing)*, Special Issue on Configuration Design, Vol. 12, No. 4, 1998.

[3] Faltings B. and Freuder E. *Configuration—Papers from the 1996 AAAI Fall Symposium*. AAAI Press Technical Report FS-96-03. AAAI Press, 1996.

[4] Pahl, G. and Beitz, W. *Koneensuunnitteluoppi*. Springer Verlag and Metalliteollisuuden Kustannus Oy, 1990. Translation of *Konstruktionslehre, Handbuch für Studium und Praxis*, Springer Verlag, 1986.

[5] Peltonen, H., Männistö, T., Soininen, T., Tiihonen, J., Martio, A. and Sulonen, R. Concepts for Modelling Configurable Products. *In Proceedings of European Conference Product Data Technology Days 1998*. Quality Marketing Services, Sandhurst, UK, 1998.

[6] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen, W. *Object-Oriented Modelling and Design*. Prentice-Hall, Inc., Eaglewood Cliffs, 1991.

[7] van Veen, E. *Modelling Product Structures by generic Bills-of-Material*. PhD thesis, Technische Universiteit Eindhoven, 1991.

[8] Snavely, G. L. and Papalambros, P. Y. Abstraction as a configuration design methodology. *Advances in Design Automation*. Volume 1, 1993.

[9] Suh N. *The Principles of Design*. Oxford University Press, New York. 1990

[10] Erens F., Verhulst K. Architectures for product families. *Proceedings of the 2nd WDK Workshop on product structuring. June 3-4 1996*. Delft University of Technology. pp. 45-60

[11] Tichem M. A Design Coordination Approach to Design for X. PhD dissertation. Delft University of Technology. 1997

[12] Andreasen M.M., Hein L. *Integrated Product Development*. IFS Publications Ltd. / Springer-Verlag, London. 1987.

Acknowledgements

We gratefully acknowledge the financial support of Technology Development Centre Finland and Helsinki Graduate School of Computer Science and Engineering (HeCSE). We thank Hannu Peltonen for his comments on how to improve the presentation of the paper. We thank Tamrock Corp. for providing us access to Ranger documentation that made the modeling effort possible.

Juha Tiihonen, Timo Soininen,
Reijo Sulonen
Helsinki University of Technology
TAI Research Centre
Product Data Management Group
P.O. Box 9555
FIN-02015 HUT, Finland
tel . +358-9-451 3242
fax. +358-9-451 4958
Juha.Tiihonen@hut.fi
Timo.Soininen@hut.fi
Reijo.Sulonen@hut.fi

Timo Lehtonen, Antti Pulkkinen,
Asko Riitahuhta
Tampere University of Technology
Machine Design Laboratory
P.O. BOX 589,
FIN-33101 Tampere, Finland
tel. +358-3-365 2627
fax. +358-3-365 2307
tle@me.tut.fi
pulkkine@me.tut.fi
aor@me.tut.fi

Appendix A: Notation