# Concepts for Modelling Configurable Products

Hannu Peltonen, Tomi Männistö, Timo Soininen,
Juha Tiihonen, Asko Martio and Reijo Sulonen

Helsinki University of Technology
TAI Research Centre and Laboratory of Information Processing Science
Product Data Management Group
P.O. Box 9555, FIN-02015 HUT, Finland

Email: Hannu.Peltonen@hut.fi

January 1998

**Abstract.** The paper defines concepts for *configurable products*, which are constructed individually for each customer order from pre-defined components according to a pre-defined *configuration model*. The concepts, which are mainly defined in terms of questions to be answered during a configuration process, include *abstract* and *concrete* component types, and the distinction between the *completeness* and *validity* of a configuration.

# 1 Introduction

Companies are becoming increasingly interested in *configurable products* as a means to satisfy a wider range of customer requirements and to reduce costs (Tiihonen et al. 1996). Unfortunately there is no general agreement even on the basic concepts. This paper describes the concepts and terms used by the Product Data Management Group at Helsinki University of Technology ('http://www.cs.hut.fi/~pdmg').[1] Instead of seeing configurable products mainly as an algorithmic challenge, we believe that the main problem is to find concepts for describing configurable products in a flexible and intuitive way. In particular the experiences of many companies demonstrate the need for tools that help the companies to maintain the constantly changing component data and configuration rules.

# 2 Overview of the Configuration Process

For us a configurable product is a product family, which consists of multiple products called the *variants* of the configurable product. The variants are constructed individually for each customer order from pre-defined components on the basis of an *order specification* and a pre-defined *configuration model*.

A product is not necessarily an end-product sold to customers. The product can also be a component within a larger product, and the customer can for example be a manufacturing or assembly unit within the same or other company. Although this paper is written in terms of physical, manufactured products, a configurable product can also be a service or other immaterial good, such as an insurance policy.

Figure 1 on the next page illustrates the process that is carried out for each customer order, leading from customer requirements to the physical products that are manufactured for the customer. Within the whole process we are mainly interested in the *configuration process*.

The input to the configuration process is an *order specification*, which is a formal representation of customer requirements. The specification is created by sales personnel, possibly in a computer-assisted sales configuration process, which may be similar to the configuration process described here.

The configuration process is controlled by a *configuration model*. The configuration model of a configurable product describes all the possible variants of the product and specifies how to create an appropriate variant for a given order specification or at least how to check whether a given variant conforms to a given specification.

The configuration process generates a *configuration*, which is a description of the specific variant that will be manufactured for the particular order. We say that the

[1] This research has been funded by the Finnish Technology Development Centre (TEKES).

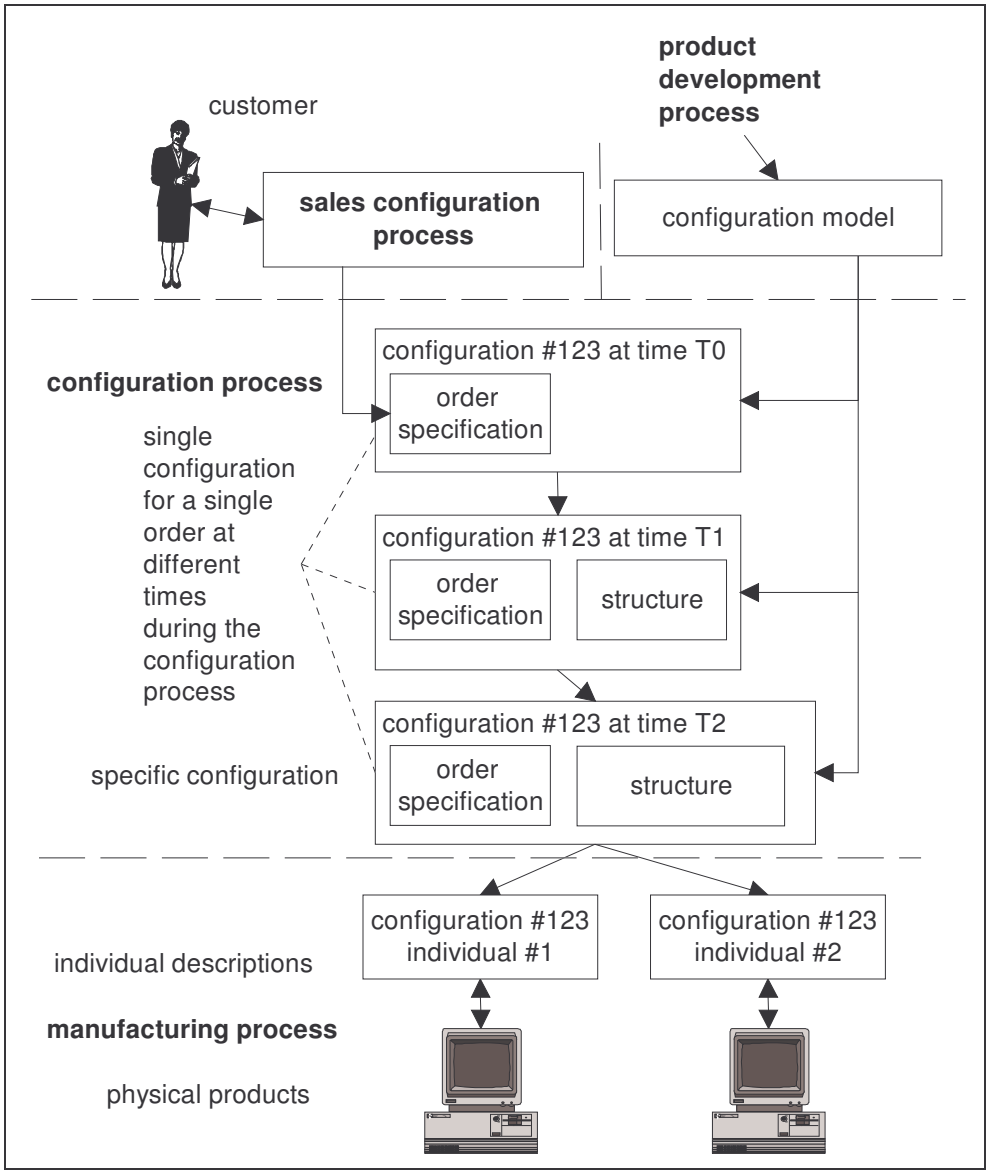


**Figure 1.** Configuration process

configuration is *based* on the configuration model that controls the configuration process. As illustrated in the figure and explained later in the text, the order specification is actually stored as part of the configuration.

A configuration does not only represent the final outcome of the configuration process but also its intermediate stages (Peltonen et al. 1994). A configuration is created at the beginning of the process, and during the process the configuration records what is known about the product variant. In general a configuration thus represents many possible product variants. During the iterative configuration process the configuration generally speaking becomes more and more detailed representing

fewer and fewer variants. Finally the configuration becomes a *specific configuration*, which represents a single variant.

Before the product is actually manufactured, one creates an *individual description* of it. Each individual description corresponds to a single *physical product* and contains additional information that is needed and created during the manufacturing process. An individual description can for example record the serial number and the maintenance history of a physical product.

# 3 Configuration Models

In contrast to some other approaches to product configuration, we believe that the description of a configurable product can often be based on a conventional hierarchical product structure. We therefore divide a configuration model into an *explicit structure* and *constraints* (Männistö, Peltonen and Sulonen 1996). Later sections show how a configuration model also contains an order specification description and procedures. The explicit structure and constraints together roughly correspond to the idea of a *generic product structure*.

## 3.1 Explicit Structure

A multilevel bill-of-material or parts list is a common presentation for many non-configurable products. To represent configurable products, the ordinary bill-of-material must be extended with optional and alternative parts and parametric components. The meanings of optional and alternative parts should be clear. A parametric component has parameters, which must be given values during the configuration process. Typical parameters include the colour or physical dimensions of a component.

The bill-of-material with optional, alternative parts and parametric components is called the *explicit structure* of a configuration model. In addition to the explicit structure, products can also be described with other constructs, such as components with ports that must be connected to other ports (Mittal and Frayman 1989).

## 3.2 Configuration Questions and Completeness

The explicit structure of a configuration model can be seen as asking questions that must be answered during a configuration process. For example, an optional part in the explicit structure asks if the component is included in the configuration; a group of alternative parts asks which alternative is chosen; and a parametric component asks what values are given to the parameters.

The explicit structure of a configuration model asks questions, and the answers are recorded in a configuration that is based on the configuration model. As the configuration gradually becomes more detailed during a configuration process, in terms of questions and answers this means that the configuration answers an increasing number of questions. A configuration that answers all the questions asked

by the configuration model is a *complete configuration*. Completeness is a necessary, though not sufficient, condition for a specific configuration. Note that all questions of an explicit structure do not necessarily have to be answered during each configuration process. Some questions are "activated" only if other questions are answered in a particular way.

We talk about questions and answers because we find them a useful metaphor for explaining completeness and other concepts related to configuration models and configurations. The explicit structure is still represented as an extended bill-of-material, not as a set of questions.

### 3.3 Constraints

The explicit structure of a configuration model defines a large number of different possible complete configurations. Typically not all of them can be accepted as specific configurations resulting from a configuration process. A complete configuration can for example be disallowed because it represents a product variant that cannot be built for technical reasons. The marketing policy of a company may also dictate that certain variants are not sold although there are no technical reasons against it. For example, a car may not be available with a leather interior in combination with a small engine.

Since all complete configurations are not acceptable as specific configurations, a specific configuration must be both complete and *valid*. A configuration model defines the validity criteria as *constraints*. Constraints are conditions, which can be evaluated in a configuration and none of which is false in a valid configuration.

The result of evaluating an individual constraint in a configuration is true, false or unknown. The result can be unknown in an incomplete configuration that does not yet answer a question that the constraint refers to. If a constraint evaluates to true or false in a particular configuration, we say that the configuration *satisfies* or *violates* the constraint, respectively. If the value is unknown, the configuration neither satisfies nor violates the constraint.

These definitions allow all combinations of completeness and validity. For example, if a configuration is incomplete and valid, all questions of the explicit structure have not yet been answered, but the answers given so far violate no constraints.

Suppose that for each question asked by the explicit structure of a configuration model there is at least one constraint that refers to the answer to this question. In this case the completeness of a configuration can also be defined in terms of constraints: a configuration is complete if no constraint evaluates to an unknown value.

## 4 Order Specifications

In this section we assume that it is possible to distinguish between an order specification, which serves as input to a configuration process, and the structure of a configuration, which represents the output from the process. This distinction is

meaningless for products that are configured without a formal order specification. Their structure is built gradually during the configuration process, and the inclusion of a particular component, for example, can sometimes be "input data" and sometimes "output data" determined by other "input data".

If, however, the product has an order specification, the relations between the specification and the configuration are also defined with constraints, which we call *implementation constraints*. For example, a computer could be associated with an implementation constraint saying that if the customer wants to run a particular application on the computer, the computer must have at least a particular amount of memory.

To write formal implementation constraints, a configuration model must describe the allowed form for order specifications that can to be used as input to configuration processes controlled by the configuration model. For example, the configuration model of a computer could specify a set of applications and say that the order specification contains some subset of this set.

We include the implementation constraints and the description of the order specification in the configuration model. The order specification itself is part of a configuration. Earlier we said that a configuration contains information for the manufacture of a product variant. Now the configuration also contains the specification to which the product is supposed to conform. Moreover, since the implementation constraints are part of the configuration model, a specific, i.e., complete and valid, configuration based on a particular configuration model represents an acceptable variant of the configurable product that also conforms to the order specification included in the configuration.

We do not assume a configuration problem to have a unique solution. There can be many specific configurations for a given configuration model and a given order specification. In this case one of the specific configurations is selected for manufacturing on the basis of some criterion, such as an optimality criterion on price or delivery time. If possible, the criterion should of course be included in the configuration model so that the model does not accept multiple specific configurations for any order specification.

We must now elaborate on the division of a configuration model into an explicit structure and constraints. A configuration model consists of an order specification description, an explicit structure and constraints. The constraints are further divided into three groups according to whether they refer to the order specification, to the explicit structure or to both.

*Specification constraints* refer to the order specification only. In other words, specification constraints define the criteria for checking the validity of an order specification that is going to be used as input in a configuration process. It is useful if the product and its configuration model can be designed so that the configuration process always results in a specific configuration for a valid order specification.

*Implementation constraints* refer to both the order specification and the explicit structure. They thus define the criteria for checking whether the structure of a given configuration implements the specification of the configuration.

*Structure constraints* refer to the explicit structure only. They define the criteria for checking whether the configuration as such represents a product that the company is able or willing to manufacture without considering any order specification.

Similarly, we must also redefine the completeness and validity of a configuration. A configuration can thus be separately complete with respect to the order specification and the structure. We can also check separately whether a configuration is valid with respect to each of the three constraint groups.

# 5 Component Types

The explicit structure of a configuration model is based on a hierarchical component breakdown structure. A product has components, which have their own components, etc. For most purposes a product as a whole can also be treated as a component. Therefore we do not distinguish between products and components, and call both simply components. In this section, however, we still ignore the breakdown structure and treat a product as a single component.

## 5.1 Component Type Generalisation

Often a company manufactures many related configurable products. Whenever the configuration models of these products have some common data, this data should be described once for all configuration models that the data applies to. Using the concepts from object-oriented modelling and from the Universal Modelling Language (UML) in particular (Fowler 1997), we represent the sharing of common data by means of *component types*, which can be arranged in a *generalisation hierarchy*.

As an example of a configurable product consider a personal computer. A computer has optional parts, such as a sound card, and alternative parts, such as different kinds of hard disks. Suppose that the computer manufacturer has two basic types of computers: a business computer and a home computer. Home computers are further available as entertainment models and educational models. We now have five related *component types* and generalisation relationships between them as illustrated in figure 2. One component type is a general description of a computer, and the other component types are more specific descriptions of the different types of computers. The general component type can for example specify that a computer must have a hard disk, and the component type for a business computer can specify that a computer of this type must have a particular kind of disk.

Component types 'Business Computer', 'Home Computer' are *subtypes* of component type 'Computer'. Component types 'Entertainment Computer' and 'Educational Computer' are in turn subtypes of 'Home Computer', and also of 'Computer'. If necessary, we can refer to the subtypes on the "next lower level" as *immediate subtypes*. For example, only 'Business Computer', 'Home Computer' are immediate subtypes of 'Computer'.

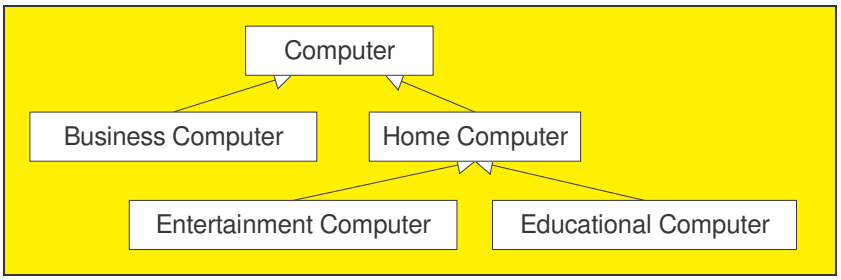


**Figure 2.** Component type generalisation

Correspondingly component types 'Computer' and 'Home Computer' are *supertypes* of component type 'Entertainment Computer', and only 'Home Computer' is the immediate supertype of 'Entertainment Computer'.

Our example shows single inheritance, in which the component types are arranged in a tree-like structure and each component type, except the topmost type, has exactly one immediate supertype. Multiple inheritance, on the other hand, allows a component type to have multiple immediate supertypes.

A component type inherits the properties of its supertypes. In other words, the order specification description, explicit structure and constraints from each supertype of a component type are "added" to those of the component type itself. For example, everything that component types 'Computer' and 'Home Computer' say about computers also applies to computers that are manufactured according to component type 'Entertainment Computer'. Multiple inheritance causes further complications, which are ignored here.

Earlier we said that each configuration is based on configuration model. We now say that a configuration model is a set of related component types and each configuration is a *component individual*, i.e., an individual of a component type.

## 5.2 Abstract and Concrete Component Types

Consider again the component type generalisation in figure 2. For the moment we assume that a product delivered to a customer must be a business computer or either of the home computer models. One cannot order simply a computer or a home computer without specifying a particular kind of computer. Component types 'Computer' and 'Home Computer' thus describe the general properties of all computers and of all home computers, respectively. They do not specify a product in "sufficient detail" in the same way as the component type for the business computer and the component types for the two specific home computer models.

To express this distinction between component types, each component type is designated either as an *abstract component type* or as a *concrete component type*. In our example, component types 'Computer' and 'Home Computer' are abstract, and the other component types are concrete. We also say that a component individual is concrete or abstract depending on whether it is an individual of a concrete or abstract component type. The "leaves" of the generalisation hierarchy, i.e., compo-

nent types without further subtypes, must be concrete. A component type with subtypes can also be concrete. According to our definition, in this case its subtypes must be concrete, too.

In terms of questions asked by an explicit structure we can say that component type 'Entertainment Computer' is concrete because the corresponding configuration model (i.e., the "sum" of the component type and its supertypes) asks all questions that need to be answered in a configuration process to make a specific product variant. The configuration model for a concrete component type also contains all necessary constraints for checking the validity of the answers. The configuration model for the abstract component type 'Home Computer' on the other hand asks "too few" questions; even when all questions have been answered, there is not enough information to serve as output from a configuration process.

### 5.3 Specific Configurations as Questions and Answers

A specific configuration must have three qualities, all of which are somehow related to the idea of a configuration being "ready": concreteness, completeness and validity. These concepts can be conveniently summarised in terms of questions that are answered during a configuration process. Note that the hierarchical breakdown structure of a product is discussed in the next section.

A *concrete component type* asks all necessary questions for a specific configuration, a *complete configuration* answers all these questions, and a *valid configuration* does not contain any "wrong" answers. Therefore a specific configuration must be concrete, complete and valid.

## 6 Components

In the previous section we introduced component types, which can be arranged in a generalisation hierarchy. A configuration, i.e., a product variant, was treated as a single component individual. In this section we consider the product breakdown structure. A configuration is now represented as a collection of component individuals arranged according to the 'has-part' relation between components.

### 6.1 Components and Parts

We say that a component has other components as *parts*. The term 'component' thus refers to an entity while the term 'part' refers to the 'part-of' relation between two components. When a component is used as a particular part, we also say that the component *realises* the part.

The parts of a component are identified with *part names*. It is important to distinguish between the name of a part and the name or other identification of a component that realises the part. For example, the parts of a car include four parts with names 'left front wheel', 'right front wheel', etc. Since all wheels of a car are

typically identical in the sense of having the same components, the four parts are usually realised with individuals of the same component type.

To illustrate the difference between a part and a component, one can regard a part of a component as a slot labelled with the part name. This slot can be filled with a component that realises the part. Note that we talk about optional parts instead of optional components. A component as such cannot be optional or obligatory. However, a part can be optional, which means that the part does not have to be realised by any component.

## 6.2 Part Refinement

As an important example of the interplay between the generalisation hierarchy and the product breakdown structure, consider figure 3, which shows component types 'car', 'bus', 'engine' and 'diesel engine'. Component types 'bus' and 'diesel engine' are subtypes of 'car' and 'engine', respectively. Component type 'car' specifies that it has a part 'engine' of component type 'engine'. Component type 'bus' is a subtype of 'car', and accordingly it can *refine* the part by specifying that the part must be of the type 'diesel engine', which is a subtype of 'engine'. In summary, each car has some kind of engine, a bus is one kind of a car, a diesel engine is one kind of an engine, and a bus has a diesel engine.

In the previous example the component type of the engine part was refined when one moved from component type 'car' to its subtype 'bus'. Parts can also be refined during a configuration process. For example, the configuration for a bus may originally specify the engine to be simply a diesel engine. Later during the process the part is refined to a subtype of the diesel engine. Typically both 'engine' and 'diesel engine' would be abstract component types. In a specific configuration the part must have been refined to a concrete component type.

The configuration as a whole is represented as a component, which has other components as parts. The "root component" of the configuration can be refined in the same way as any of its parts. This means that a configuration that is originally an individual of one component type can change during a configuration process to be an individual of a subtype of the original component type.

Above we described the refinement of a part from a component type to one of the subtypes. A part can also be replaced with an individual of a supertype, which is an example of "backtracking" during a configuration process.

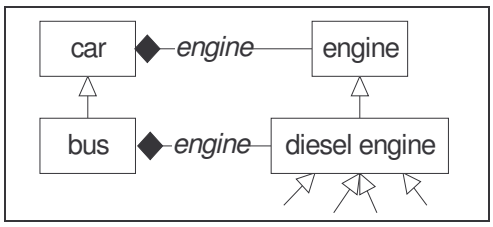


**Figure 3.** Part refinement

### 6.3 Local and Global Properties

Each component in the configuration has its own validity constraints according to its type. When we say that a specific configuration must be valid, we obviously mean that no component in the configuration is allowed to violate its own constraints.

We therefore distinguish between the *local* and *global* validity of a component. Our earlier definition of validity corresponds to local validity. A component is thus *locally valid* when it does not violate any of its own constraints (i.e., constraints defined in the component type and in its generalisations). A component is *globally valid* when the component itself and all its parts are locally valid. Here the parts of a component include the whole product hierarchy, i.e., the components that realise the "immediate" parts of the component as well as the components that realise the parts of these components, etc. Similar distinction between local and global properties can also be made for the completeness and concreteness of a component.

Our final definition for a specific configuration is thus as follows: a globally concrete, globally complete and globally valid component individual.

## 7 Validation and Generation

We have introduced constraints as a mechanism for specifying the validity conditions for a configuration. Nevertheless, our goal is to build a software tool that helps a user to create a specific configuration for a particular configuration model and order specification. This tool should obviously do more than just check the validity of a configuration that the user has somehow created.

Some constraints can be solved automatically, i.e., given a set of constraints, one can automatically derive a procedure that generates a configuration that satisfies the constraints (Shah and Mäntylä 1995, sec. 8.3.3). Nevertheless, we believe that many constraints in actual configurable products are too complex to make constraint programming a general solution to configuration problems.

We make a distinction between the *validation* and *generation* of a configuration. A configuration is validated with constraints, and generated with *procedures*. A procedure can employ various techniques, including conventional procedural language, logic programming and, in some cases, constraint programming. A procedure can also be an external program, which is started by the configuration tool with input values derived from the answers already available in the configuration and which adds new answers to the configuration.

Sometimes it is difficult to separate validation and generation. For example, suppose part of the configuration is generated with an external stress analysis program. It seems impossible to associate this procedure with a more meaningful validation constraint than one that checks that the current output values have been computed from the current input values with the correct program.

# References


Fowler, Martin. 1997. *UML distilled: Applying the standard object modeling language*. Addison-Wesley.

Männistö, Tomi, Hannu Peltonen, and Reijo Sulonen. 1996. View to product configuration knowledge modelling and evolution. In *Configuration—papers from the 1996 AAAI Fall Symposium*, ed. Boi Faltings and Eugene Freuder, 111–118. AAAI Press.

Mittal, Sanjay, and Felix Frayman. 1989. Towards a generic model of configuration tasks. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence (IJCAI)*, 1395–1401.

Peltonen, Hannu, Tomi Männistö, Kari Alho, and Reijo Sulonen. 1994. Product configurations—An application for prototype object approach. In *The 8th European Conference on Object-Oriented Programming (ECOOP '94)*, ed. Mario Tokoro and Remo Pareschi, 513–534. Springer-Verlag.

Shah, Jami J., and Martti Mäntylä. 1995. *Parametric and feature-based CAD/CAM*. John Wiley & Sons.

Tiihonen, Juha, Timo Soininen, Tomi Männistö, and Reijo Sulonen. 1996. State-of-the-practice in product configuration—A survey of 10 cases in the Finnish industry. In *Knowledge intensive CAD*, Vol. 1, ed. Tetsuo Tomiyama, Martti Mäntylä, and Susan Finger, 95–114. Chapman & Hall.