View to Product Configuration Knowledge Modelling and Evolution

Tomi Männistö, Hannu Peltonen and Reijo Sulonen

Helsinki University of Technology
IIA Research Centre and Laboratory of Information Processing Science
Otakaari 1, FIN-02150 Espoo, FINLAND
{Tomi.Mannisto, Hannu.Peltonen, Reijo.Sulonen}@hut.fi

Abstract

Knowledge management has been a major problem in the implemented configuration systems. One reason for the difficulties is the complexity of the product descriptions in these systems. The product experts can no longer understand product the way it is described. The problem is addressed in the object-oriented spirit by looking for ways of making the configuration product structure models more comprehensible. A data model for describing generic product structures is defined. It forms a basis for modelling the evolution of both product configuration knowledge and delivered products.

Introduction

Many knowledge intensive systems have had problems with knowledge maintenance. Product configuration systems are no exception in this respect, e.g., XCON (Barker & O'Connor 1989, McDermott 1993). There is no universal agreement on the knowledge needed in product configurators, let alone on the representation of this knowledge. We therefore find it impossible to discuss configuration knowledge management without reference to what kind of knowledge is being managed.

The basic issue in configuration modelling is a mechanism for describing structures of multiple product variants within a single data model. Many different methods have been used for modelling this generality in product descriptions (Schönsleben & Oldenkott 1992, van Veen 1991, Cunis et al. 1989, Mittal & Frayman 1989 and Heinrich & Jungst 1991). We identify two basic product structure modelling methods in these approaches, which we refer to as *explicit* and *implicit methods*. We then define a data model which uses both these methods in combination.

There is no single way of looking at products of a company. Products can be classified according to various criteria. A classification criterion suitable for one purpose may turn out to be awkward in another context. We try specifically to avoid this kind of problem by constructing our model explicitly for product configuration purposes.

After definition of the model we discuss the configuration knowledge management problem in the environment where the model would be used. Finally we outline our vision for a research agenda for the evolution of configuration models.

This paper reports ongoing work of the Product Data Management Group at Helsinki University of Technology.

Short Definitions of Basic Terms

A *physical product* is manufactured according to a *specific product structure*. A specific product structure is a collection of component descriptions organised in a *part-of hierarchy*. Each *component description* contains the necessary information for making or ordering a component. A specific product structure is sometimes called a *bill-of-materials* (BOM) or a *configuration*.

In many companies a product can be varied according to customer requirements and therefore actually refers to a set of similar products. This set can be described with a *generic product structure*, which corresponds to a number of different specific product structures. In a generic product structure a *generic component reference* refers to a set of alternative components.

A *configuration process* starts with a generic product structure and customer requirements. Customer requirements are mapped to a *technical specification*, which describes the customer needs in the language understood in the configuration process. The goal of the configuration process is to find a suitable (valid, complete and possibly optimal) specific product structure that is among the alternatives described by the generic product structure. We do not assume any particular kind of configuration process, it may be fully automatic or rely partly or entirely on decisions made by a human.

A product can be described from many different points of view, each potentially defining a different product structure. In this paper, the generic product structure, or *configuration model* as it is sometimes called, describes only those aspects of the product that are relevant during a configuration process.

Modelling Generic Product Structures

There are several possible ways of modelling generality in product structures. We divide them here roughly in *explicit* and *implicit product structure description methods*. The terms *explicit* and *implicit* indicate how directly the generic product structure identifies the components used. We use terms *explicit* or *implicit model* for a data model that is based on an explicit or implicit product structure description method, respectively. A single generic product structure can also employ both kinds of methods.

We use the term 'product structure' here, although all implicit methods do not aim at describing the organisation of the components in a part-of hierarchy. They may connect the components in some other way than in a part-of hierarchy or perhaps describe only the component set and assume the structure to be well known or fixed.

In an explicit method the generic product structure is described by stating the components, their organisation in the part-of hierarchy and possible choices for them. For example, *a MountainBikeX has part frame which is either MBFrameX or MBFrameY*.

In an implicit method, the description consists of knowledge about the compatibility of components, connectivity of components or other constraints. For example, in description of a display unit, the component 'monitor' may state that it needs a high resolution video signal, and a particular graphics card may state that it provides exactly that kind of signal. A T-connector may state that it takes one video input signal and provides two outputs of the same signal. There is no explicit description of the components of a working display unit. If two monitors are needed for a certain system, one graphics card and a T-connector with two monitors will do the job.

It is also possible to use both explicit and implicit methods in a single model. For example, on the explicit side one may state that a product P has components *a* and *b*, where *a* is either A1 or A2 and *b* is either B1 or B2. Implicit knowledge in B2 may then say that it is incompatible with A1 and they should, therefore, not be used in the same product.

In explicit models it is typically straightforward to enumerate all possible specific product structures. Finding the desired structure may still take too much time if all the possibilities need to be investigated.

In a purely explicit model a selection between alternative components can be made without considering other component selections. In practice, however, the alternatives in some components depend on each other. Sometimes these dependencies can be solved with a *global context*, which is established at the beginning of the configuration process.

For example, suppose a product has component 'motor' with alternatives '220V' and '110V', and component 'power switch' with analogous alternatives. The global context for this product could include attribute 'operating voltage'. After the context has been established, the motor and the switch can be chosen with explicit rules. Alternatively one could use implicit rules such as: *if the product has a 110V (220V) motor, it must have a 110V (220V) switch*, or alternatively *a 110V (220V) motor cannot be combined with a 220V (110V) switch.*

Explicit model thus more directly tells what components must be chosen for a valid configuration whereas implicit model defines conditions that must be satisfied by valid configurations.

We believe that explicit product structures are in many cases easier to construct and understand than implicit ones. Nevertheless, many products include validity conditions that require implicit methods. The data model that is outlined later in the paper therefore has an explicit basis that is extended with implicit constraints.

In the following we give some examples of both explicit and implicit product structure description methods.

Sample Explicit Methods

Set of BOMs. This is a trivial case and not actually a real generic product structure model. The method is, however, widely used in the industry for modelling even large numbers of variants.

Generic, Variant and Parametric BOMs. These define alternatives for certain components of a BOM and possibly global variables for determining the choice. (Schönsleben & Oldenkott 1992, van Veen 1991)

AND-OR Graphs. At each level of an AND-OR graph a node breaks down to either its components (AND) or to one selection between choices (OR). The AND and OR levels alternate, so that an AND level is followed by an OR level and vice versa.

Combined Classification and Component Hierarchies. This is a sophisticated version of an AND-OR graph as a combination of component and classification hierarchies. There each non-leave node can be either divided into its components (just like AND) or it can be specialised to its subclass (a kind of an OR). This approach is used in SAP Configurator (SAP 1994) and is a close relative of the PLAKON model (Cunis et al. 1989).

Composite Instance Variable. Composite instance variables are used in object-oriented models for representing components (Kim, Bertino, & Garza 1989). Such a variable represents actually a set of components if its domain is a class that has subclasses; any of its subclass being an eligible component.

Sample Implicit Methods

If-then Rules. In many expert systems if-then rules can be used for implicitly describing product structure. For example, *IF component A OR B is in configuration THEN component X must be included, too.*

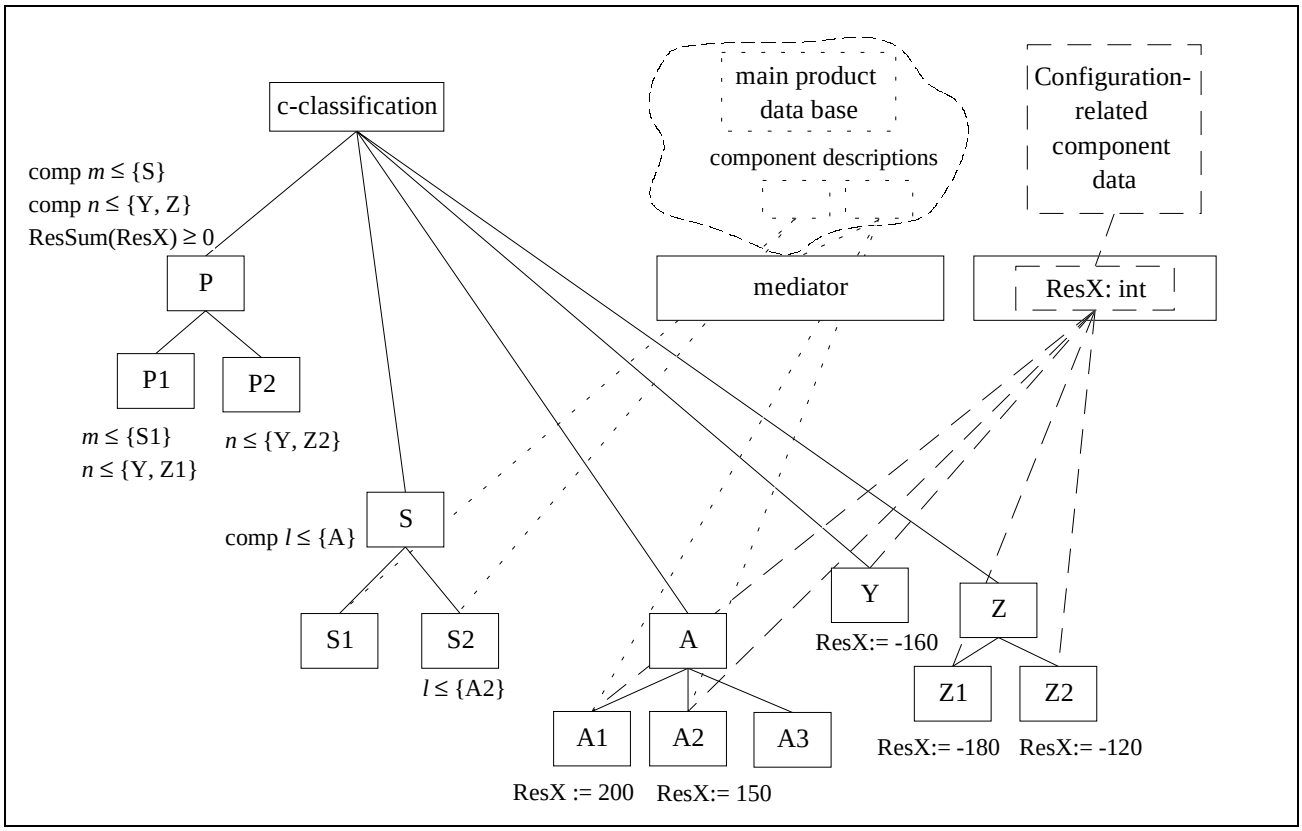

Figure 1. Example generic product structure in the model.

Incompatibility and Compatibility Constraints. One way of stating the possible uses of components is to state which components can or cannot be with other components in a configuration. These facts can be recorded as n-ary tuples.

Interfaces, Ports and Connectors. These provide deeper modelling for compatibility of components. The basic idea is that in a configuration certain components need, can or cannot be connected. This is modelled, for example, by ports and connectors that must fit together in a valid configuration (Mittal & Frayman 1989).

Resource-based Modelling. In resource based generic product structure modelling the idea is that some components *consume* something, such as, power, fuel, ventilation, etc., while certain components *supply* these resources. In a valid specific product structure all resources need to be in balance. For example, the total produced net power must be more than maximum consumption. (Heinrich & Jungst 1991)

Generic Product Structure Model

This section gives an overview description of the properties of a data model. It is based on our earlier work (Peltonen et al. 1994). The model includes both implicit and explicit product structure description methods. The explicit description defines a superset of all valid specific product structures and implicit methods are used to prune out the invalid cases. In the following we first give an example of a generic product structure using the model and then explain the concepts in detail.

An Example of a Generic Product Structure in the Model

Generic product structure description of P in Figure 1 declares that P has two components *m* and *n*. Component *m* should be an S, while *n* should be either Y or Z. This is an explicit product structure description. These definitions are refined in subclasses P1 and P2. Component S is a generic product structure of its own; it has one component *l*. Finally, Y and subclasses of A and Z have no components. In the figure, solid line represents is-a relationship; component relationships are not represented graphically.

As an example of implicit constraints we use here resources. In Figure 1 there is an example resource ResX. It is a resource that is supplied by A1 and A2 and consumed by Y, Z1 and Z2. Suppliers and consumers are linked to the resource by dashed lines. These classes may assign a value to ResX; a negative value meaning consumption. A balance constraint for the resource is defined in product P; it states that the components of P cannot consume resource ResX more than they supply.

The explicit product structure limits the possible component combinations, no other components but the declared ones are allowed. Therefore, one can enumerate

all the possible components for the tuple (*m*, *n*, *l*) in P2 according to the explicit product structure:

(S1, Y, A1), (S1, Y, A2), (S1, Y, A3),
(S1, Z2, A1), (S1, Z2, A2), (S1, Z2, A3),
(S2, Y, A2),
(S2, Z2, A2).

The implicit resource constraint then prunes out the ones where ResX consumption is greater than production. We assume that ResSum in P returns zero if ResX is not found in a component. The only valid combinations in P2 for the component 3-tuple (*m*, *n*, *l*) are:

(S1, Y, A1),
(S1, Z2, A1), (S1, Z2, A2),
(S2, Z2, A2).

After enumeration, each possible specific product structure of a configuration model can be checked against applicable constraints, including technical specification. This provides a trivial method for finding a solution for particular customer requirements. The existence of the trivial method only shows that a solution, if one exists, can be found; the method is hardly useful for a real configuration process. More efficient heuristics is needed in a practical application.

Main Concepts of Data Model

Classification Hierarchy. Our model is based on classes. They are organised into a classification hierarchy. The properties of a class are inherited along the classification hierarchy.

A large set of components cannot be managed without organising it somehow. Classification is a useful tool in product modelling; it provides means for organising, but one must decide on the classification criteria. Components can be classified, for example, according to the functions they perform or the types of the manufacturing processes producing them.

We therefore require that a classification of component descriptions is created specifically for configuration modelling. We call this classification hierarchy *c-classification*.

Although, we here discuss only one c-classification, a company may require different c-classifications for different product families. This means that same components may have different classification and even different properties in different product families. These components should perhaps be somehow related or shared between the classifications. This is, however, an issue we will not address in this paper.

The c-classification has one particular property that we call *component subtyping*. Component subtyping states that subclasses of a class C may always be used in place of C in component descriptions of a generic product structure. For example, if class MBFrame has subclasses MBFrameX and MBFrameY, a generic component reference ‘MountainBikeX has part MBFrame’ means that a MountainBikeX may have either MBFrameX or MBFrameY as its frame.

Generic Component References. In basic object-oriented product models components are represented by instance variables, which are used for referring to the components. The domain of an instance variable is a class. If the class has subclasses, the instance variable may also take as its value an instance of any of the subclasses (Banerjee et al. 1987). This is a property known as *subtyping*.

Subtyping is a generic concept and, as such, has nothing to do with components. For example, if A1 and A2 are subtypes of A, this alone does not imply that they are valid as component representations in place of A. Therefore we have a more specific subtyping concept for a classification where classes represent components. That is what we call component subtyping. Component subtyping is not a property of any classification by accident. A classification hierarchy, the c-classification in this case, must be specifically constructed, with the help of product experts, so that it is valid with respect to component subtyping.

Component subtyping defines a one form of generality for component descriptions—a component is not necessarily from the domain class, since also the subclasses provide valid choices. Generic product structure descriptions, however, also need a more general mechanism for defining component alternatives. For example, assume a class A that has three subclasses A1, A2 and A3. For one product A1 and A2 are valid component alternatives, while for another A2 and A3 are the ones. There are no classes that could directly be used as component domains for these products, nor is there a simple way of defining them.

In our model a generic component reference is described as an instance variable and its domain as a non-empty set of classes. This means that the value of the variable must be a reference to one of the listed classes or their descendants. If an ancestor of a class is also in the domain, the class itself is redundant and can therefore be removed. For these domains we use a special notation: $c \leq \{A, B\}$. This is a short for a constraint $\{c \leq A \vee c \leq B\}$, where $c \leq A$ means that c is a component instance from class A or its decsendant.

One particular property we want to have for generic component references is *refinement*. A subclass can *refine* a component attribute by making its domain more specific than in the superclass. More precisely, there are two operations: 1) domain set may be replaced by its proper subset and 2) in place of a class one can write its subclasses. Both operations may be applied multiple times. This allows definition of an abstract component reference which may be refined in subclasses into more concrete descriptions.

For example, component c in class C may have set {A, B} as its component domain. This means that c must be either an A or a B. If A has subclasses A1, A2 and A3, the possible refinements for the domain of c in the subclasses of C include {A} and {A1, A2, B}. Refinement of generic component references is present in only some

generic product structure models, e.g., in PLAKON (Cunis et al. 1989).

Other Component Data

The c-classification is not be suitable for modelling all the properties needed in configuration. Two classes that share common properties may be placed far apart from each other and cannot therefore inherit the common properties along the c-classification. This is a consequence of strictly enforcing component subtyping in the construction of the c-classification. There is also another, perhaps even more important, reason why all component data cannot be defined in the c-classification. Component data is used widely in a company, and in this picture configuration is only one player. One cannot assume that all component data needed in a configuration process would also be primarily modelled and maintained in the c-classification.

For these reasons we include in the model two other sources for component data*: main product data base* and *additional configuration related data*.

A main product data base represents here all the component data that is maintained elsewhere in a company. It need not be a single data base and its detailed structure is not of great importance here. The main issue is that certain data items can be imported from the main product data base to the c-classification. In Figure 1 importing of properties from the main product data base is illustrated by dotted lines.

For coupling the main product data base and the c-classification a mediator is introduced between the two. This mediator provides properties for the classes in the c-classification; a class may refer to a property that is visible in the mediator. Such reference is effectively the same as if the property was declared in the class. As a difference, however, the relation to the origin of the property in the main product data base is maintained.

Additional configuration related data defines properties that are tightly connected with the c-classification, but do not have a single place there. As an example a resource is an implicit product structure modelling method that cannot always be easily expressed as a property of a single class in the c-classification. This is because suppliers and consumers of a resource are not typically subclasses of the same parent class in the c-classification; for instance, components that either supply or consume electric current may be quite different.

Resources serve here only as an example of additional configuration related data that cannot be nicely inherited in the c-classification, they should not be taken as a key characteristic of the model. Resources may be modelled on top of the explicit product structure as additional data items that can be imported in the c-classification. This way a resource can be defined in single place and then related to arbitrary component descriptions in the c-classification. In Figure 1 this is shown by dashed lines.

In the balance constraint sum of resources in the components of a particular specific product structure is used. The sum is calculated by a special function ResSum, which goes through all the components starting from the one for which the balance constraint is defined. For each component it checks whether the resource is visible in the component and a value is assigned to it. If so, it adds the assigned value to the sum and otherwise nothing.

Evaluation of a balance constraint involves recursive transversal of all components of a specific product structure. Evaluation as such is easy for a known specific product structure. Computational problems may, however, arise for a mechanism that tries to find an appropriate structure with a given initial constraints, i.e., a technical specification. We do not address here the problem of satisfying a balance constraint or fixing it were it false. In fact we assume the same as for the whole model; the model only describes a set of valid specific product structures, not how to find a particular one in a configuration process.

Evolution in Generic Product Structures—A Research Agenda

This section gives some guidelines for our future work on configuration knowledge management.

Our special interest is to understand concepts needed to describe the evolution of products and related computer representations. In principle, we can distinguish between two different kinds of change process. On one hand, during their life-time the physical products as real world objects are modified for a number of reasons: they are being serviced, their functionality may be upgraded, faulty components are being replaced by correct ones, etc.

Modifications are in many cases made not by the organisation that made the products but by the customer itself or by some more or less independent service or maintenance company. Due to the increasing importance of after sales activities and long life-cycles of products it has become increasingly more important to maintain consistent representation of product *somewhere*.

On the other hand, the generic product structures have life of their own. The owner company introduces new products, enhances existing models, replaces components with newer ones due to the number of different reasons. On some industries, such as electronics and information technology, these renewal processes are quite hectic.

Both these change processes are difficult as such. We have seen only few examples of systems capable of maintaining accurate representations of delivered one-of-a-kind products. The maintenance of the generic product structures without exploding their complexity needs powerful tools and policies which guarantee that they remain manageable during their life-cycle. Unfortunately these two processes are not, at least in theory, always

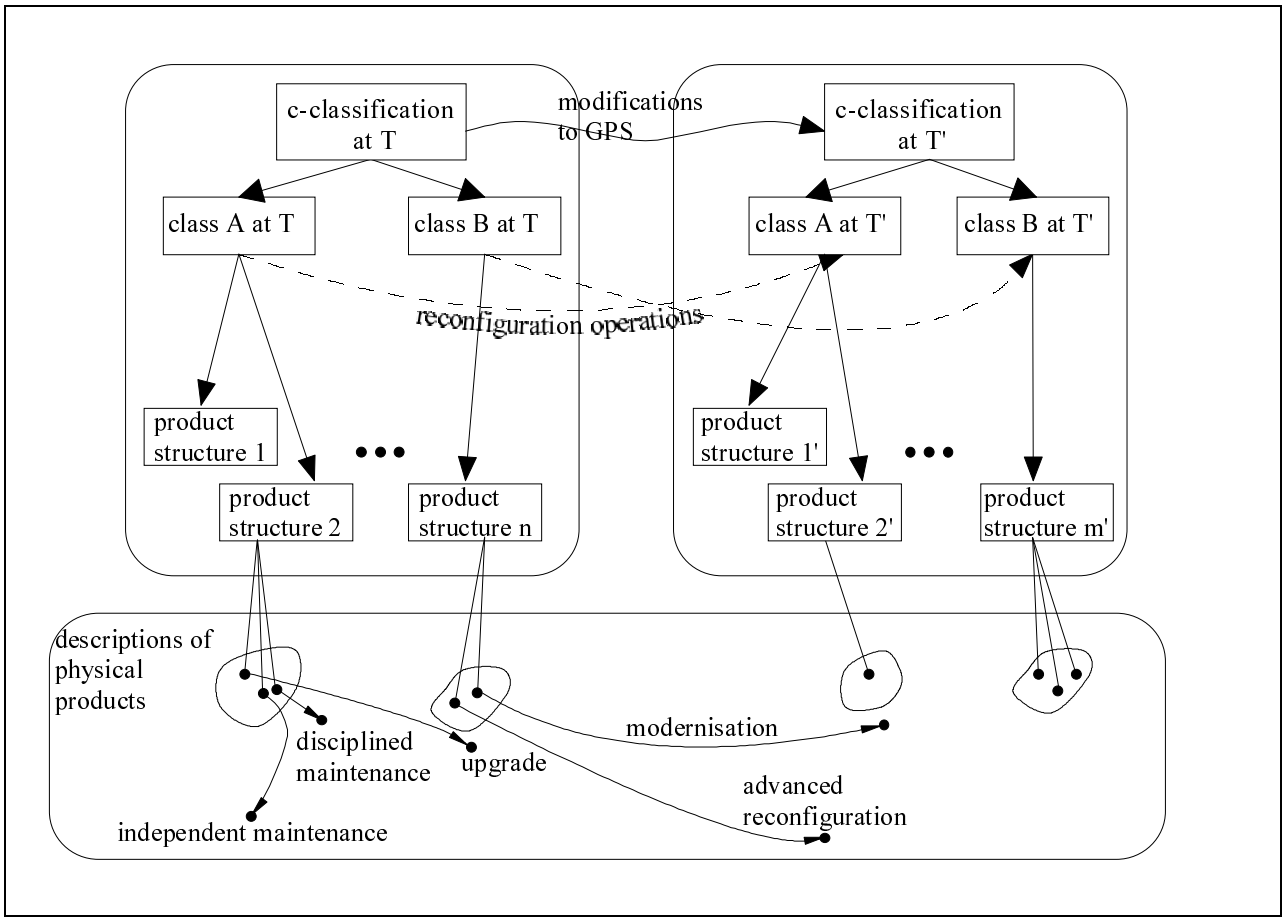

Figure 2. Evolution of Generic Product Structure and Physical Products.

independent. A product designed and manufactured long time ago under the presumptions of the generic product structure effective at that time may be brought back to the factory years later for functional upgrade. How to relate the old product structures, i.e., physical and generic, to the currently used generic product structure? We would like to look for conceptual framework which might cope with problems of this nature at least in certain conditions.

Evolution of Physical Products

By physical products we mean the products that have been manufactured and delivered to the customers. In the following we discuss different kinds of modification operations on physical products.

Figure 2 shows generic product structure, i.e., a c-classification, at times T and T'. At the bottom of the figure there is a rounded box for descriptions of physical products. There a dot in a roughly shaped circle represents a physical product delivered to a customer. Each "circle" encloses descriptions of physical products related to one specific product structure. Modifications that change the structure of a physical product are illustrated by arrows which move a description away from its original specific product structure. Some modified descriptions are still close to a "circle". This suggests similarity, e.g., functional equivalence, to the products of the related specific product structure.

Independent Maintenance. These include typical on-field operations. One modifies the product according to the need without considering the relation to generic product structure or other information. It is very difficult for a product management system to support this kind of operations in any other way than by recording modifications and current state of physical products. At any rate, the description of a physical product should be updated to reflect the modified physical product.

Disciplined Maintenance. A company may define service kits that do not modify the functionality of the products but replace, for example, certain parts that may suffer from ageing. If individual components are changed, a company may, for example, have the policy that each new component version must be compatible with older ones, i.e., any old component version can be replaced by a newer version. The resulting new specific product structure is close to the original with a known difference.

Upgrade or Modernisation. These are true reconfiguration operations, where the functionality of the product is changed. An upgrade enhances the functionality of a product, for example, by increasing certain maximum capacity. A modernisation brings an old physical product to the level of functionality of a newer product generation. One problem is that upgraded and modernised products are different from the new configured ones, even if they have the same functionality. That is because the modified products consist of a mixture of old and new components, while new products are made from new components only. An upgraded product becomes typically close to another product that offers higher capacity and a modernised product has the functionality of a product of a later generation, typically the current generation at the time of the modernisation.

Advanced Reconfiguration. A configuration process creates a specific product structure for a customer specification. Similarly, an advanced reconfiguration process can produce a new specific product structure starting from an old physical product. In general, advanced reconfiguration can produce an entirely different product from any previously designed one.

Evolution of Generic Product Structures

We distinguish between three maturity levels for configuration management environments on the road towards a dream environment. A first level environment has the capability of describing configuration knowledge in a way that it can be maintained. A second level environment stores, in addition, the full history of both generic product structures and descriptions of physical products. Then, a third level environment, i.e., a dream environment, also includes knowledge about the nature of modifications. One would know in detail how two generic product structures differ and be able to operate with product structures of different times, e.g., upgrade old physical products using new components.

Currently most configuration systems struggle with the problems related to the first level. Knowledge management is still a real problem in configuration systems. Much of the problem stems from the complexity of the configuration models—they are “programmed” by computer specialists and not understood by the product experts. One of the goals we hope to achieve is a structured and understandable method for describing configuration knowledge.

Modifications to the generic product structure result from various sources. These include market situation, customer demands, different regulations in different market areas, advances in technology, corrections of old designs, just to mention few. A company must have a clear policy how these changes are incorporated to the configuration knowledge. Definition of good practices is a part of the overall solution.

Management of changes is essentially based on decisions, typically made by humans. These decisions need to be recorded and they can be used for describing the relevant history of the whole model.

It is not clear which concepts are needed for storing the evolution of generic product structures. A straightforward way is to create a copy of the whole configuration model after each modification. In this much of the information on changes is lost. For a better modelling of the evolution one should record the changes of smaller pieces of information. Classes, for example, could be changed by creating new versions of them. The problems arise then how class versions relate in classification hierarchy and how to incorporate versions in generic product structures.

In addition, the history of changes to the descriptions of physical products needs to be retained. The real challenge is to incorporate the evolution of generic and physical product structures in a useful manner.

For example, imagine the following scenario. *A customer has two products from the company: one three and the other five years old. Last year a capacity upgrade was made for the newer one. Now the customer wants to know whether a similar upgrade would be possible for the older one as well.*

Given the full historical data, one can find out the product configuration environment, e.g., configuration model and components, at the time the upgrade of the newer product took place, and naturally the current situation. In addition, there would also be a full history of service modifications.

Typically, the answer to the customer inquiry would be positive, but usually an engineer is required to design the needed modifications almost from scratch. With an ideal environment the engineer might find the following possibility: *There is an modernisation package for the old product to the level X and from there a modernisation package to the current level where the requested capacity upgrade is directly possible with newest components. These two modernisations can be combined, but one must manually check the versions of components q, s and l for compatibility.*

Conclusions

Product configuration modelling is essentially based on description of generality in product structures. We divided the used methods two categories: explicit and implicit and discussed briefly how previous approaches relate to these. Our own approach uses both methods: a generic product structure description has an explicit structure as basis and implicit rules are used for pruning out invalid structures. We believe that this would make the generic product descriptions more clearer and therefore easier to construct and maintain.

The management of evolution of configuration knowledge is a critical long term goal for a configuration modelling environment. We approached the problem by describing the key elements in the state of the practice and then painted a vision for the future work. This is an area

with plenty of open questions; many of them fundamentally difficult. We try to emphasis the needs of the industry as criterion for selecting the course of future research in the field of product data modelling and management.

Acknowledgements

This research has been partly funded by the Academy of Finland.

References

Banerjee, J.; Kim, W.; Kim, H.-J.; and Henry, F. K. 1987. Semantics and implementation of schema evolution in object-oriented databases. In Proceedings of the International Conference on Management of Data (SIGMOD).

Barker, V. E., and O'Connor, D. E. 1989. Expert systems for configuration at Digital: XCON and beyond. *Communications of the ACM* 32(3):298-318.

Cunis, R.; Günter, A.; Syska, I.; Peters, H.; and Bode, H. 1989. PLAKON—an approach to domain-independent construction. In Proceedings of the Second International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems IEA/AIE-89.

Heinrich, M., and Jungst, E. W. 1991. A resource-based paradigm for the configuring of technical systems from modular components. In Proceedings of the seventh IEEE conference on Artificial Intelligence Applications.

Kim, W.; Bertino, E.; and Garza, J. F. 1989. Composite objects revisited. In Proceedings of the International Conference on Management of Data (SIGMOD).

McDermott, J. 1993. R1 ("XCON") at age 12: Lessons from an elementary school achiever. *Artificial Intelligence*, 59(1-2):241-247.

Mittal, S., and Frayman, F. 1989. Towards a generic model of configuration tasks. In Proceedings of IJCAI 89.

Peltonen, H.; Männistö, T.; Alho, K.; and Sulonen, R. 1994. Product configuration—an application for prototype object approach. In Mario Tokoro and Remo Pareschi, editors, Object Oriented Programming, 8th European Conference, ECOOP'94. Springer-Verlag.

SAP 1994. Der SAP Konfigurator (Reference Manual).

Schönsleben, P., and Oldenkott, H. 1992. Enlarging CAD and interfaces between PPC and CAD to respond to product configuration requirements. In Pels H. J., and Wortmann, J. C., eds., Integration in Production Management Systems.: Elsevier Science Publishers B.V.

van Veen, E. A., 1991. Modelling Product Structures by generic Bills-of-Material. Ph.D. diss., Technische Universiteit Eindhoven.