Open Data Modelling Issues in Product Configuration

Detailed abstract

Tomi Männistö, Hannu Peltonen and Reijo Sulonen
Department of Computer Science
Helsinki University of Technology
Otakaari 1, FIN-02150 Espoo, Finland
{Tomi.Mannisto, Hannu.Peltonen, Reijo.Sulonen}@hut.fi

1 Introduction

There is an increasing demand for individually tailored products with short delivery times. This demand can be satisfied with *configurable products,* which can be easily customized according to customer specifications. The designers of a configurable product must devise a plan that can be used repeatedly for realizing customer orders as working combinations of the available components. We use the term *configuration model* for this information. The concept is similar to some uses of the term *product family.* After a customer has ordered a configurable product, the order and a product configuration model are used for creating a *product configuration,* which describes a single instance of the product.

1.1 DREAM CONFIGURATION ENVIRONMENT

A dream product configuration environment would provide a nice way for modelling configurable products, support the configuration process and cope beautifully with the evolution of things in it. In the following we discuss the most important concepts it needs. The very basic requirement is the capability of modelling the entities manipulated in the environment. The state-of-the-art of data modelling does not provide that.

As a basis *classification* with inheritance are needed. These are rather well understood and generally agreed upon.

Next one is *part-of relationship,* which is a well-known concept for instances of data. Although it has a large number of possible semantics, we believe that one can agree on useful semantics for a specific application, so this is not the main point here. The point is to treat the part-of relationship as a first-class property with respect to inheritance and refinement. The following two examples try to clarify this.

First, "*car has engine; and lorry (which is a car) has diesel engine (which is an engine)*". This is a part-of relationship between classes and a refinement of it in a subclass.

Second, "*a lorry of type L1 has an engine of* ***either type E1 or E2 but not E3*** *(all subclasses of diesel engine)*". Here the property of class L1 refers to neither one nor all the subclasses of diesel engine. Modelling this by separate descriptions for each different product structure is not a solution when the total number of potential product variants increases to thousands or millions.

The third concept, *versioning*, has a large variety of possible semantics. It can be used for describing many things from historical evolution to unspecified similarity of entities. Most of the previous work deals with versions of instances. With classes the main consideration has been in the evolution of database schemas and the conversion of instances to reflect the modified schema. This work, however, is not directly suitable for modelling the evolution of product descriptions. For example, if a company changes the description of its products, say, by replacing a component with a newer one, none of the delivered product instances is going to change automatically. Therefore, instances cannot be converted to reflect the modification in product description as is typically done in database schema-evolution.

To sum up, data modelling for the dream environment requires: classification and inheritance; component relationships between classes with support for refinement in subclasses and complex domains; and versions of classes with clear distinction between evolution of general product descriptions and product instance descriptions and with richer versioning semantics than just evolution.

1.2 MAIN RESULTS

Using configurable products as a real life modelling case, this paper shows that the current understanding of the principal data modelling concepts: classification, component relationship and versions is not adequate. The most problematic issues are confronted when the concepts are needed in combination. By describing the requirements of the dream system we point out some limitations of the current approaches.

We have expressed our goals in a form of a dream, since we cannot possibly claim to have solved them. By analysis of the underlying principals we believe we have achieved guidelines for treating the three concepts in combination.

We also present an initial methodology combining the three concepts. Although not entirely satisfactory, it is meaningful from the viewpoint of product configuration modelling.

1.3 SIGNIFICANCE AND RELEVANCE OF MAIN RESULTS

Our work is motivated by existing problems in manufacturing industry. The companies use different techniques to cope with these problems [Veen91]. In many cases, however, the means do not scale up when the number of product variants increases. The problem of modelling products with a large number of variants is not going to go away, quite contrary—better customer satisfaction is becoming increasingly important; large variety of choices is a part of it.

Class versioning and component relationships are not unknown to the data model researchers. These concepts are so central in data modelling that we have no doubt they will be discussed and probably solved within the next ten years. We have raised important aspects to these issues. Our contribution clarifies the concepts and shows a place for a different kind of schema modification than known from traditional databases.

2 Problem Domain

In the following we discuss some relevant points in the use of the classifications, components and versions for configuration modelling or product modelling in general.

2.1 PRODUCT MODELLING

Product modelling must consider three different kinds of entities:

- *physical products* (instances)
- *descriptions of physical products* (instances)
- *general product descriptions* which describe sets of potential physical products (instances), e.g., products that can be manufactured.

In the real world there are only physical products. The other two entities exist only as abstract concepts or data, which somehow try to represent physical products (already existing products or ones to be manufactured).

The description of a physical product is bound to a specific physical product instance delivered to a customer. A general product description is used for producing new physical products. It does not, as such, represent any particular physical product instance. Typically, a description for a physical product instance is generated from a general product description.

2.2 CLASSIFICATION

Classification is a method for organizing modelled entities in a hierarchy[1] consisting of linked nodes called *classes*. *Subnodes* of a node can be referred to as *subclasses* or *specializations* of the node, and its *parent node* as *superclass* or *generalization* of it. Classification relationship is also called *is-a relationship*.

A class can represent a set of *existing instances*, e.g., delivered products, or a set of *potential instances*, e.g., that describe also products to be manufactured. In addition, an instance can be instantiated from a class, or it can belong to a class just because it has the right properties.

A class is a more general description of its subclasses, and the subclasses of a class share the properties of the class. The primary reason for classifying entities can be to maximize the amount of shared properties. This is in a way like looking at the properties of different things and organizing them then according to similarities in these properties. Classification can also be done regardless of the common properties. Whether entities share properties is then a separate issue; some properties may need to be duplicated.

We want to emphasize the many different ways of interpreting and constructing a classification hierarchy for a given problem domain with set of (potential) instances. It is not the question of constructing just a classification hierarchy, but finding an appropriate one.

2.3 COMPONENT RELATIONSHIP

This section is divided according to the categorization of different kinds of product modelling entities in Section 2.1. First we mention two general issues.

1. We omit here the discussion of multiple classification parents.

First, component structures are inherently recursive. A thing can be divided in components (or parts) and the same method can be applied to each of these components. In product modelling, this recursion is typically stopped at certain level of detail. Second, the distinction between terms *product* and *component* is relative and context dependent—what is a product in one sense may be a component in other.

Component Relationship of a Physical Product

There are certain laws on the relationship between parts and the whole of a physical product [Kim89, Halp92, Liu92]:

- Each physical product instance is unique, i.e., any two physical product instances have separate identities.
- One physical component cannot be a part of two physical wholes, e.g., one car motor cannot be a motor of two cars (at one time, at least).
- A physical component cannot be a component (even recursively) of itself.
- Deletion of a whole physical product implies also deletion of its components.

Component Relationship in Description of a Physical Product Instance

A data model suitable for describing a physical product instance needs the semantics reflecting the properties of the physical products as outlined above. These cannot however be stated as general requirements of part-of relationship, since there are cases where they do not apply. For example, one paragraph can be a part in multiple documents [Halp92].

The component structure of a product depends also on the viewpoint. STEP gives examples of different product definition contexts such as conceptualized product structure, functional structure and as-planned-for-manufacture [ISO92]. In other words, a product has multiple component structures; in this paper we do not discuss relationships between them, but assume that only one of them is considered at a time independently of others.

In addition, concepts such as inheritance via component links may need to be considered in a model. For example, the colour of a car can be inherited by its component door, or power of the engine inherited by the whole car. [Chan90, Katz90, Halp92, Liu92]

Component Relationship in General Product Description

The component relationship changes its nature when we move from the description of a single physical product instance to the description of a set of product instances. For example, one component (as an abstract entity—not a physical instance!) can be used as a part in product P1 as well as in product P2 when P1 and P2 are general descriptions of two kinds of products.

When modelling component relationships in general product descriptions one may need to observe the following special requirements:

- *Refinement of component relationship*. E.g., lift has a machinery, and in model lift L1A machinery M1A is used.
- *Complex domain specification*. E.g., type of the machinery of a Lift type L1 is *either M1A or M1B.*
- *Optional component*. A configuration may, but does not have to, include an optional component.
- *Number of similar components*. For example, a lift may require similar doors for each floor.

2.4 CHANGE MANAGEMENT

The distinction between descriptions of physical products and general product descriptions is relevant also while considering the evolution of products.

Product Changes in Industry

A product instance starts its own life outside the company after it has been delivered to a customer. In many cases the company cannot just forget the product instances; it needs to provide a range of after-sales services. These include maintenance, upgrades, modernization, etc.

General product descriptions are, on the other hand, quite different; is not a light operation for a company to change them. Therefore, these changes are often restricted to few product releases a year. These product release procedures differ from actual design processes and clearly from after-sales operations.

Change as Data Modelling Concept

Versioning is a concept that is used for describing the evolution of products [Katz90]. In addition, versions can be used for various other purposes such as concurrency control, recovery, performance enhancement and update-free databases [Ditt88]. Therefore, no single semantics for versioning exists. In the following are some of the most common ones [Feil91].

- *History.* Versions can represent historical development of designs.
- *Experimental path of development.* In a design process multiple versions may be generated as alternative solutions.
- *Variants.* One product can have two slightly different variants, for example, for different market areas.
- *Concurrent development.* Separate versions can be used for enabling concurrent modifications of a design.

There can be two kinds of component references to versions: *statically bound* and *dynamically bound* [Kim87]. A statically bound reference specifies explicitly a component and one of its versions. Allowing only statically bound references has severe limitations: rebinding of references must be considered each time a new version is introduced, and in case of multiple variant component structures, different references are needed for each variant.

Therefore, typically some dynamic mechanism is introduced. One concept for giving a dynamic referencing point to a set of versions is *generic instance* [Kim87]. A reference to a generic instance is eventually bound to one of the versions.

A dynamic component reference to a specific version can be bound at various points in time. Should this be done when the reference is created for the first time, or a design is approved or rather when it is used in manufacturing? This is an issue that needs to be decided upon.

Generic version concept makes the rather strong assumption that all versions of a component are similar or compatible with regard to the component references. A more general approach would allow more specific descriptions of subsets of versions of a component. Each such subset would bear one kind of semantics from the component use viewpoint.

2.5 SIMPLE EXAMPLE

This section shows by an example what the combination of the three modelling concepts means. Although it seems rather trivial, it has not been solved; no product data modelling environment adequately combines the three concepts.

Suppose we are modelling product Lift and one of its components Machinery. In Figure 1 class Machinery represents the set of all possible machineries the company wants to represent with this model. M1 then represents a subset of that set; all M1 type machineries.

To combine component relationship and classification hierarchy we want to say that each Lift has a component Machinery. This is a general high level description of the component relationship between Lift and Machinery. L1, as a specialization of Lift, may then refine this relationship. In general, a component relationship may involve an expression which describes a selection constraint, e.g., the component must be *either M1 or M2A*.

For modelling the evolution of classification hierarchy, we assume that each class has a set of versions. Each version of a class can, in principle, be combined with any version of its parent class. This construction creates potentially a large number of definitions and some of them may not even make any sense. Therefore, a mechanism for controlling the legal classifications paths is needed.

Of all the component relationships only some are presented in Figure 1. For example, version v1 of L1 specifies its component by exp1, which uses references to generic version M1 and to version v1 of M2A. In L1A exp2 refines exp1 by having references to M1A v1 and M2A v1.

3 Towards Realization of the Dream

3.1 PUTTING CLASSIFICATION AND COMPONENTS TOGETHER

In this section we look at some product data modelling approaches that combine classification and part-of relationship.

Component Relationship between Data Instances

Traditional object data models contain classes and instances.[1] A simple way of representing product structures is to have component relationships only between data instances [Kim87, Kim89].

1. One should not be confused by the two kinds of instances: product instances and their descriptions vs. data instances and data descriptions (classes). These two kinds of instances do not necessarily coincide, since a data instance can also be used for representing a set of products, not only a single product instance.

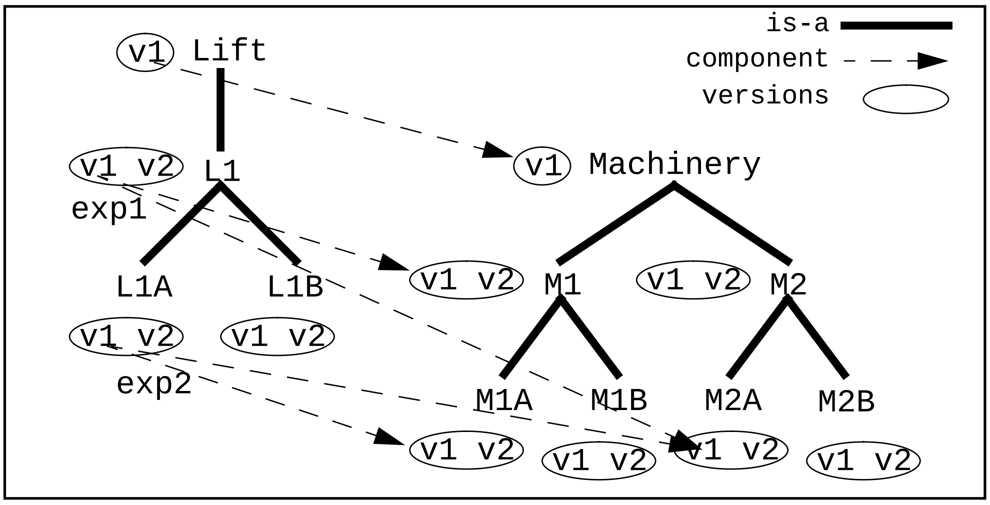

Figure 1: Simple example.

This approach is useful for representing physical product instances when the creation of the product structure for each instance is a non-routine operation, i.e., when there is no predefined product structure. This is the case, for example, in innovative design where a designer is trying to construct an acceptable product structure for the first time. Anyway, classification is between classes while component relationships are between instances; they are, therefore, quite separate in the approach.

Data Instance Templates for Product Instance Descriptions

This might be a bit confusing case. Here instances of the data model are used for representing both sets of physical product instances and individual physical product instances. One has a template object for each different product structure [Eren94]. This template is then used for creating a description for a particular product instance.

This approach provides a pragmatic solution for many cases. It is, however, not a good solution when the number of product variants increases. One would end up with a large number of templates and management of them would not be easy in the long run.

Conceptual Component as Class Property

A more object-oriented approach would be to define component relationships as properties of classes (e.g., as attributes of classes). An example of this is shown in Figure 2, where class MountainBike has component *frame* as a reference attribute (denoted by word **compref**) pointing to class Frame3.

In such model the domain of a reference attribute is typically restricted to a single class (with its subclasses). So it would not be possible to express the fact that in MountainBikes wheels W1 and W3 are possible choices, but not W2. Therefore, separate classes MB001 and MB002 are needed in Figure 2. This is not a good solution unless the number of possible component structures is small.

Also the refinement of component relationships is missing. This would be possible in an object model where an attribute can be refined in subclasses by restricting its domain.

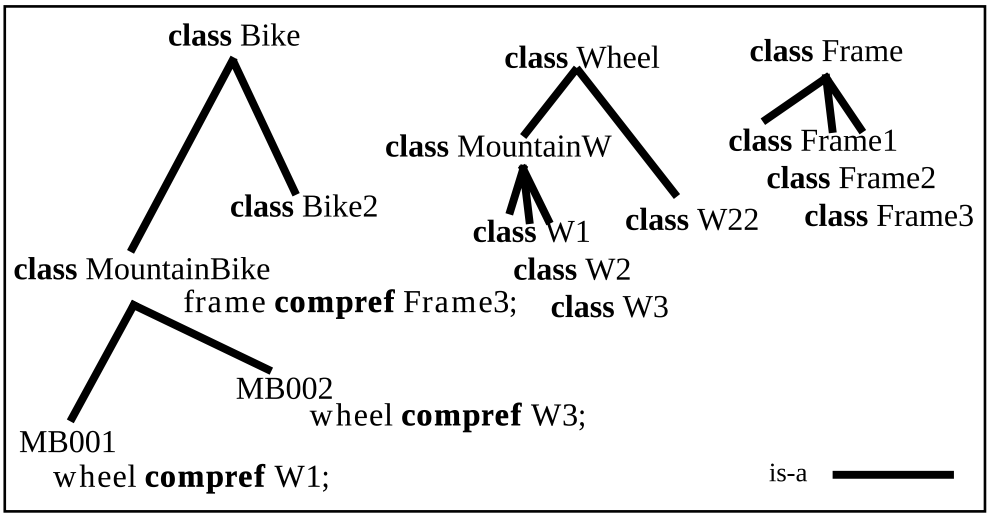

Figure 2: Component relationship as an attribute of a class.

PLAKON Model

The model in Figure 2 can also be presented in a single connected graph by drawing a (different kind of) line from each **compref** to the respective domain class. The PLAKON model uses this kind of technique and comes close to the dream model by supporting also refinement of component relationships [Cuni89]. It does not, however, support as a primary concept the complex domains for component relationships, e.g., in Figure 2 MountainBike wheels W1 or W3. Therefore, there is still the same problem with a large number slightly different structures.

Representing Conceptual Components by Constraints

This section introduces very briefly the model we have in mind [Pelt94]. We prefer the use of special attributes and constraints for modelling conceptual component relationships. Also in [Kram91] constraints are used for selecting components.

In Figure 3 component attribute is declared with word *comp* and its domain specified by a constraint in curly braces. Expression $a \leq B$ is used for stating that the value of a should refer to B or its subnode. The constraints are inherited via is-a links, and a node should satisfy all its (including inherited) constraints in order to be valid. Therefore, a refinement of a component relationship is achieved simply by declaring an additional constraint in a subnode.

In this model we have also a special concept for describing a physical component relationship. Important point is the explicit separation between conceptual and physical component relationships (in other words, at class level and instance level, respectively).

This is roughly what is needed for component relationships in a data model suitable for the purpose of realizing our dream. Introduction of class versions to the model has naturally also reflections here.

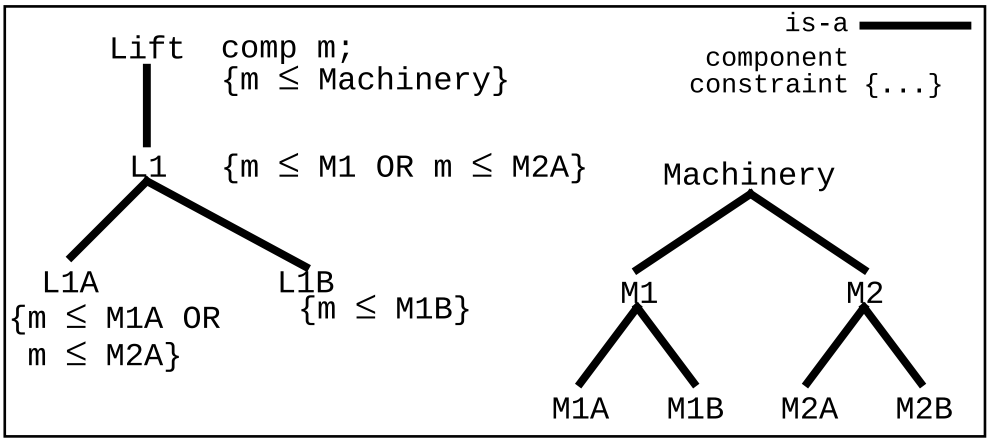

Figure 3: modelling conceptual component relationship by constraints.

3.2 Evolution of Classes

Database Schema Evolution

Changes to the schema of a database can hardly ever be avoided. This makes schema evolution a relevant topic of database research [Skar86, Bane87, Penn87, Skar88, Clam92].

A principal assumption in database schema-evolution is that instances can be converted. This assumption is based on the principle that schema constantly describes the same (or at least almost the same) real word entities. Schema modifications just slightly alter the way those entities are represented.

A configuration model, however, represents the potential instances that are being sold and manufactured. When this description is modified it does not represent the same configuration instances from a different angle, but an entirely new set of possible configurations. The old configuration instances are not represented by that schema and, therefore, conversion is not meaningful.

This shows the basic difference in the treatment of schema in traditional databases and in configuration modelling. The principal concepts look similar in both approaches and advances in either one are probably beneficial also for the other one. Nevertheless, they are targeted at different problems.

Evolution of Configuration Models

One must separate the evolution of general product descriptions from the evolution of product instances (and their descriptions). In the following we try to outline a very simple solution for the straightforward evolution of configuration models. We do not discuss here the evolution of configuration instances.

We assume that each class has a set of versions. In addition we require that all versions of a class must be “semantically compatible”, i.e., similar in a certain, human interpreted, sense. Now, in principle, a class version can inherit from any version of the parent class. This leads to a large set of possible inheritance paths. Some of them do not make sense semantically and some may be even syntactically invalid, e.g., include contradicting constraints or dangling references to attributes. Therefore, a mechanism for describing meaningful inheritance paths in the classification hierarchy is needed.

By defining the latest version of a class always to be the current version, we get a (trivial) mechanism for restricting the possible inheritance paths. There is now exactly one possible definition (i.e., the current version) for a class at any given time. So, if the time of each modification, i.e., class version creation, is recorded, we can reconstruct the hierarchy at any given time in the past. This, of course, assumes that in one modification transaction all the necessary modules are changed so that the current versions always form a consistent classification hierarchy.

Although this model is trivial, it corresponds rather well to the evolution of classes in configuration models. It allows the reconstruction of the configuration environment for a past situation, a property that is mostly missing from current product modelling methods, although the reconstruction is very much needed for after-sales operations. Forgetting the old schemas in product modelling leads to loss of valuable information.

Generic references are no problem, if we assume that an instance can live only at one point in time. It is, now, not possible to create a version at a certain time and then bind its dynamic references at another time. In other words, references must be bound according to the schema of the creation time of an instance.

4 Conclusions and Future Work

We introduced an initial model which has the required properties for component relationships between classes. The moral is that the proper treatment of component relationships in general product descriptions needs attention; use of an object-oriented data model as such does not solve product modelling problems.

Class versioning is clearly an open issue. Primary requirement is that versioning should be supported at all levels of classification hierarchy as outlined in Figure 1.

We have touched on some of the important issues for adding class versions to a model with component relationship between classes. They are summarized here.

- Definition of the legal inheritance paths.
- Generic references to class versions, especially conceptual component references. What a referring end can assume from the referred end?
- Treatment of instances. This is different from schema-evolution of databases, but some conversion operations might be needed. For configurable products these are special after-sales operations called reconfiguration.

We introduced a trivial solution to the first problem and used generic instance concept for the second one. These are too simple solutions to be satisfactory.

The next step towards the model required by the dream is to introduce concept for describing similarity among class versions. We are working on a concept we call *interface*. Interfaces are used for capturing similarities of class versions by defining subsets of versions. In the most generic case this would be done by explicitly specifying which versions belong to the interface. Interface mechanism would also allow modelling of richer versioning semantics, provides a more specific point of reference for class versions and introduces a more elaborate mechanism for specifying the legal inheritance paths.

5 References

[Bane87] Jay Banerjee, Won Kim, Hyoung-Joo Kim, and Henry F. Korth. Semantics and implementation of schema evolution in object-oriented databases. In *Proc. of the International Conference on Management of Data (SIGMOD)*, pages 311–322, 1987.

[Bato85] D.S. Batory and Won Kim. modelling concepts for VLSI CAD objects. *ACM Transactions on Database Systems*, 10(3):322–346, September 1985.

[Chan90] Ellis E. Chang and Randy H. Katz. Inheritance in computer-aided design databases: Semantics and implementation issues. *Computer-Aided Design*, 22(8):489–499, October 1990.

[Clam92] Stewart M. Clamen. Type evolution and instance adaptation. Technical Report CMU-CS-92-133, Carnegie Mellon University, School of Computer Science, CMU, Pittsburgh, PA 15213-3891, June 1992.

[Cuni89] R. Cunis, A. Günter, I. Syska, H. Peters, and H. Bode. PLAKON—an approach to domain-independent construction. In *The Second International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems IEA/AIE-89*, volume 2, pages 866–874. ACM, 1989.

[Ditt88] Klaus R. Dittrich and Raymond A. Lorie. Version support for engineering database systems. *IEEE Transactions on Software Engineering*, 14(4):429–436, April 1988.

[Eren94] Frederic Erens, Alison McKay, and Susan Bloor. Product modelling using multiple levels of abstraction—instances as types. *Computers in Industry*, (24):17–28, 1994.

[Feil91] Peter H. Feiler. Configuration management models in commercial environments. Technical Report CMU / SEI-91-TR-7, Software Engineering Institute, 1991.

[Halp92] Michael Halper, James Geller, and Yehoshua Perl. An OODB "Part" relationship model. In *Proc. of the Information and Knowledge Management (CIKM-92)*, pages 602–611, 1992.

[ISO92] Product Data Representation and Exhange —Part 203. Application Protocol: Configuration Controlled Design. ISO CD 10303-203, 1992.

[Katz90] Randy H. Katz. Toward a unified framework for version modelling in engineering databases. *ACM Computing Surveys*, 22(4):375–408, December 1990.

[Kim87] Won Kim, Jay Banerjee, and Hong-Tai Chou. Composite object support in an object-oriented database system. In *Conference on Object-Oriented Programming Systems and Languages (OOPSLA)*, pages 118–125. ACM, October 1987.

[Kim89] Won Kim, Elisa Bertino, and Jorge F. Garza. Composite objects revisited. In *Proc. of the International Conference on Management of Data (SIGMOD)*, pages 337–347. ACM, 1989.

[Kram91] B.M. Kramer. Knowledge-based configuration of computer systems using hierarchical partial choice. In Third International Conference on tools for Artificial Intelligence TAI 91, pages 368–375. IEEE, 1991.

[Liu92] Ling Liu. Exploring semantics in aggregation hierarchies for object-oriented databases. In *IEEE International Conference on Data Engineering*, pages 116–125, 1992.

[Pelt94] Hannu Peltonen, Tomi Männistö, Kari Alho, and Reijo Sulonen. Product configurations—an application for prototype object approach. In Mario Tokoro and Remo Pareschi, editors, *Object Oriented Programming, 8th European Conference, ECOOP'94*, pages 513–534. Springer-Verlag, 1994.

[Penn87] Jason D. Penney and Jacob Stein. Class modification in the GemStone object-oriented DBMS. In *Conference on Object-Oriented Programming Systems and Languages (OOPSLA)*, pages 111–117. ACM, 1987.

[Skar86] Andrea H. Skarra and Stanley B. Zdonik. The management of changing types in an object-oriented database. In *Conference on Object-Oriented Programming Systems and Languages (OOPSLA)*, pages 483–491, 1986.

[Skar88] Andrea H. Skarra and Stanley B. Zdonik. Type evolution in an object-oriented database. In B. Shiver and P. Wegner, editors, *Directions in Object-Oriented Programming*, pages 393–415. MIT Press, 1988.

[Veen91] Eelco Anthonie van Veen. *Modelling Product Structures by generic Bills-of-Material*. PhD thesis, Technische Universiteit Eindhoven, 1991.