Männistö T, **A Conceptual Modelling Approach to Product Families and their Evolution.** Acta Polytechnical Scandinavica, Mathematics and Computing Series. No. 106, Espoo 2000, 166 p. Published by the Finnish Academies of Technology. ISBN 951-666-543-8. ISSN 1456-9418.

Keywords: Product families, product data modelling, product evolution

Abstract

Many industries exhibit a growing trend towards better customisation. This means a larger product variety, which can be implemented in the form of product families. The main idea of product families is to combine the economy of scale offered by mass-products with the high degree of adaptation to customer needs provided by project products, which necessitates a mechanism for capturing the variation in an effective manner. In this thesis, the focus is on the conceptual modelling of product families, particularly on the problems related to the evolution of product family descriptions and the product individuals created according to them. The main goal of this thesis is to find means for representing product families in information systems so that their evolution is adequately catered for.

Investigation of the STEP standard, namely its part AP214 for the automotive industry, leads to observation of *conceptual mismatch*. That is, a need for new conceptual modelling methods, which cannot be provided directly by traditional databases or current PDM systems. A conceptual platform with a set of invariants was defined to capture the semantics for modelling product families and their individuals. Three detailed studies were conducted on the relation of schema and individuals of product families. The first detailed study concentrated on the instantiation process, the second on the modelling support for reconfiguration of product individuals and the third on elaborate modelling of product individuals with multiple levels of abstraction, which enables systematic support for the long lifetime and large variety of product individuals.

To Satu

Supervisor

Professor Reijo Sulonen
Department of Computer Science and Engineering
Helsinki University of Technology
Espoo, Finland

Reviewers

Professor Pasi Tyrväinen
Department of Computer Science and Information Systems
University of Jyväskylä
Jyväskylä, Finland

Dr. Tech. Jouko Vuoskoski
IT Manager
Aker Finnyards
Rauma, Finland

Opponents

Professor Kaj A. Jørgensen
Department of Production
Aalborg University
Aalborg, Denmark

Professor Pasi Tyrväinen
Department of Computer Science and Information Systems
University of Jyväskylä
Jyväskylä, Finland

Contents

List of Original Contributions

This thesis is based on the following original contributions, which are referred to in the text by their Roman numerals (I–V):

I Männistö T, Peltonen H, Martio A, Sulonen R. Modelling Generic Product Structures in STEP. Computer-Aided Design 1998;30(14):1111–8.

II Peltonen H, Männistö T, Alho K, Sulonen R. Product Configuration—an Application for Prototype Object Approach. In: Tokoro M, Pareschi R, editors. ECOOP 94. Proceedings of the 8th European Conference on Object-Oriented Programming; Bologna, Italy. Springer-Verlag; 1994. p. 515–34.

III Männistö T, Sulonen R. Evolution of Schema and Individuals of Configurable Products. In: Chen PP, Embley DW, Kouloumdjian J, Liddle SW, Roddick F, editors. Advances in Conceptual Modelling, Lecture Notes in Computer Science 1727; Springer; 1999. p. 12–23.

IV Männistö T, Soininen T, Tiihonen J, Sulonen R. Framework and Conceptual Model for Reconfiguration. In: Faltings B, Freuder E, Friedrich G, Felfernig A, editors. Papers from the AAAI Workshop on Configuration; AAAI Press; 1999. p. 59–64.

V Männistö T, Peltonen H, Soininen T, Sulonen R. Multiple Abstraction Levels in Modelling Product Structures. Technical report. Laboratory of Information Processing Science, Department of Computer Science, Helsinki University of Technology; 2000; Report no.: TKO-B148.

1 Introduction

In many industrial sectors, the large product variety is becoming commonplace. Customers are no longer happy with "having the product in any colour as long as it is black". Operation on wider markets requires adaptation to different environments, regulations and cultures. Thus, the trend in many industrial sectors is increasingly towards better customisation leading to larger product variety, which can be implemented as *product families*. Large product variety, however, does not come free. It complicates sales and pricing, increases and imbalances inventories, confuses manufacturing processes, just to mention a few problems. It also makes product data management more difficult. Instead of single products, the company needs to manage data on product families comprising possibly millions of variants. This thesis concentrates on the problems in data management of product families.

Figure 1 shows the industrial environment for the problem area of this thesis. The order-delivery process is illustrated by transparent arrows whereas, grey arrows show (partially) the information flow of product descriptions. Product families require a selection of one variant for a specific customer need. This selection process is in the figure referred to as "Sales Configuration". The selection of the variant is based on product family descriptions, which are developed and maintained with the help of a *product data management* (PDM) system (see e.g., Miller et al. 1997, Peltonen 2000) by the process labelled as "Design" in the figure. In practice, the knowledge management of a sales configurator and PDM system may be separate, which easily causes problems of consistency. Therefore, in this thesis it is envisioned that changes to product families are made in the PDM system and propagated to the sales configurator. After the sales configuration, manufacturing information about the product individual must be generated. This *bill-of-materials* (BOM) or *order-BOM* is the structure of a product individual to be manufactured and contains details irrelevant to the variant selection.

Traditionally, PDM systems have been rather weak in dealing with product families, although better systems have emerged recently. In this thesis, the management of product family information and the generation of order-BOM after the sales configuration are envisioned to be within the scope of a PDM system. To distinguish from traditional BOM-based PDM systems, such an imaginary PDM system is called an *advanced PDM system*. An advanced PDM system has an extended scope in

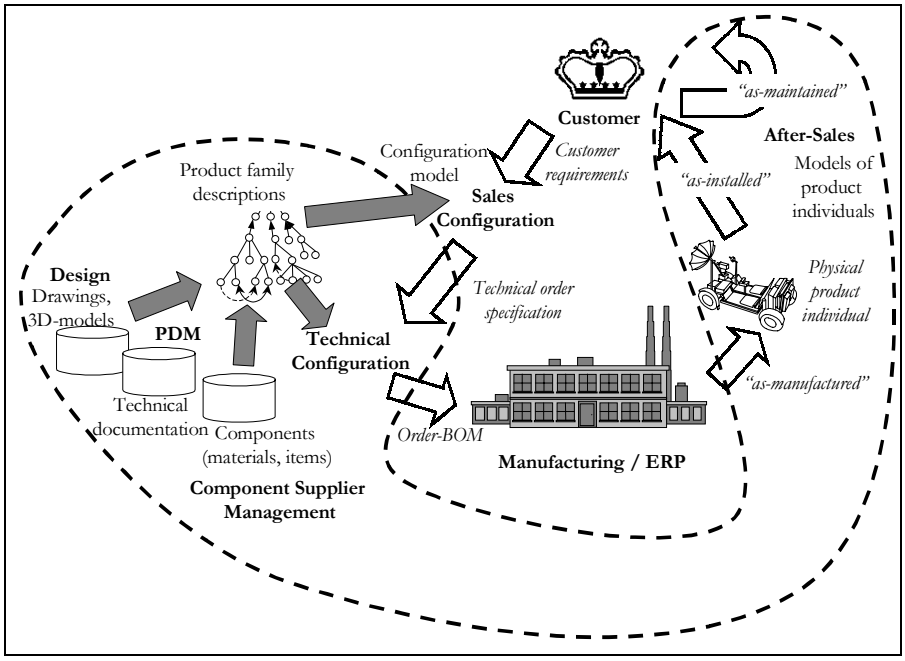

Figure 1. Industrial environment for the problem area of the thesis.

managing the evolution of product families and the individuals created according to them. Therefore, the scope of an advanced PDM system also includes the evolution of product family individuals after they have been delivered to a customer, i.e., after-sales operations. The scope is illustrated in Figure 1 by a dotted line. Out of scope, as discussed in this thesis, is still manufacturing, which is catered for by Enterprise Resource Planning (ERP) systems. An advanced PDM system is thus an imaginary system for which this thesis develops conceptual grounds for modelling and managing the evolution of product families and their individuals.

In other words, this thesis addresses the conceptual modelling of *product families;* in particular when a product family contains a large number of variants. The idea in product family modelling is to capture the potentiality of different structural variants of a product within a single data representation. This thesis concentrates on certain fundamental aspects in modelling product families and the characteristics of the evolution of product families. In the latter, the two main evolution processes relevant to product families must be distinguished as follows.

First, as companies develop their products, product family descriptions are constantly changed. Product family descriptions conceptually correspond to a traditional schema. However, in traditional database schema evolution, the existing data is typically converted to reflect changes in the schema. In product management, this is not the case—old product individuals are typically not affected when a company changes its product families. The evolution of the schema in product family modelling, therefore, significantly differs from traditional schema evolution.

Second, product individuals may have long lifetimes and histories of their own. More importantly, they do evolve independently of the schema since as a product individual is modified in after-sales, the change typically involves new components from schemas newer than the one used for creating the individual. Consequently, the modified product individual consists of old and new components and thus typically conforms to no schema version; that is, such a product individual could never have been sold as a new product individual.

As the above points illustrate, the modelling needs of product families crucially differ from the data modelling in traditional databases because the evolution of individuals is not reflected in the schema and, consequently, the individuals need not conform to the schema. Therefore, for example, it is necessary to relax the strict conformance of individuals to the schema in modelling the evolution related to product families.

1.1 Research Method

This thesis is a methodological research which builds on the conceptual analysis of a configuration task that has been developed within the Product Data Management Group (PDMG) of Helsinki University of Technology (Tiihonen 1994, Soininen 1996, Tiihonen et al. 1996, Peltonen et al. 1998, Soininen 1998, Soininen et al. 1998, Tiihonen et al. 1998, Tiihonen 1999) and continues the earlier work of the author on the evolution of product families (Männistö et al. 1993, Männistö et al. 1995, Männistö et al. 1996, Männistö 1998). This thesis concentrates on the conceptual modelling of product families and their evolution, which includes the conceptual modelling of product families, the conceptual modelling of evolution of product families and the conceptual modelling of evolution of individuals of product families.

The work analysed the conceptual and ontological foundations of product family modelling based on experiences in Finnish industry. From this analysis, a framework for addressing the evolution of product family descriptions and individuals were constructed. Within this framework, three detailed studies were conducted in specific problem areas identified in the industry.

1.2 Results

The main contribution of this work is in clarifying the conceptual complexity of large product variety, in particular with respect to evolution. The results are in more detail as follows.

The thesis clearly articulates the differences, i.e., the conceptual mismatch, between the properties of traditional data modelling methods and the needs of product family modelling. The focus of this thesis is not on the actual concepts for modelling product families, as these have been investigated elsewhere. The focus is on the fundamental aspects of product family modelling as the problems in modelling product families with traditional data modelling methods stem from them. The fundamental aspects are investigated mainly to understand and explain the differences in the role of schema and individuals, and consequently to motivate the proposed treatment of the schema and individuals in capturing the evolution of product families. The fundamental problems in modelling product families were clearly seen when the largest standardisation effort in product data modelling, namely the STEP standard, was investigated and its approach to modelling product families was shown inadequate (I).

A conceptual framework that characterises the evolution of product families and provides ontological foundations for supporting their evolution in information systems was developed (III). The conceptual framework addresses in more detail the relation of the schema and individuals of product families. Product individuals are created according to the schema. Thereafter, the schema and individuals are more independent than their counterparts in databases, but they are not entirely independent. The complicated relation of schema and individuals is further investigated in three detailed studies.

One promising non-traditional approach for modelling product families was investigated by developing a classless object model for describing product families and supporting the instantiation process of product individuals, i.e., configuration process, as object refinements (II).

The second detailed study (IV) addressed reconfiguration. Reconfiguration is a significant area of after-sales, especially for companies with product families. Individuals of product families are in many cases investments with a long lifetime and are modernised at some point in their lives. In reconfiguration, an existing product individual is modified (i.e., reconfigured) to meet new requirements. In the constructed approach, reconfiguration is modelled by reconfiguration operations, which capture the valid conversions of old product individuals. The selected approach is based on the experiences gathered from two dozen Finnish industrial companies that manufacture configurable products.

The third detailed study (V) concentrated on the evolution of individuals of configurable products as experienced by one industrial company. It defined a novel approach for modelling product individuals that do not conform to any version of the schema used for creating the individuals. The approach provides multiple abstraction levels for representing the individuals by a specialisation hierarchy of models.

In summary, the results of this thesis establish a conceptual basis for the product data management of product families in the manner hardly any of the current PDM systems have begun to address. It, however, is not assumed that an advanced PDM system would be a monolithic system working with uniformly accepted concepts. On the contrary, the concepts of the system should be adjustable, which essentially means that product data management policies can be selected from predefined options without a need for customisation by laborious programming.

2 Review of the Literature

This work is influenced by many scientific disciplines including computer science, product design and industrial management. More specifically, it includes areas, such as conceptual modelling, object-oriented databases, design databases, temporal databases, schema evolution of databases, product data management and product configuration modelling. The purpose of this section, however, is not to provide an extensive review of all these fields. Instead, the goal is to familiarise the reader with the problem area and to help in understanding the contents of this thesis.

2.1 Conceptual Modelling

Information modelling is concerned with construction of computer-based symbol structures that model some part of the real world (Mylopoulos 1998). In other words, the objective in all data and information modelling is to describe a *universe of discourse* (*UoD*). The representation of the UoD should allow an accurate expression of the user's conception of a relevant portion of the world (Eastman and Fereshetian 1994). It has also been acknowledged that evolving product data requires conceptual models that are independent of the software and hardware environments in which the models are implemented (McKay et al. 1996).

A *conceptual information model* or *conceptual model* offers semantic terms for modelling an application. The terms may include, for example, Entity, Activity and Agent (Mylopoulos 1998). Conceptual models are distinguished from *physical models* and *logical models,* which describe the structure of the database in terms of a database management systems and the implementation of a database in the secondary memory, respectively (Batini et al. 1992, Jørgensen and Raunsbaek 1998). The most important components of conceptual modelling for this thesis are *static ontology* and *dynamic ontology* (Theodoulidis and Loucopoulos 1991, Mylopoulos 1998).

Within the static ontology, much of the power of conceptual modelling comes from *abstraction mechanisms,* of which the most important ones for this thesis are *classification, generalisation* and *aggregation* (Rumbaugh et al. 1991, Batini et al. 1992, Fowler 1997, Mylopoulos 1998).

- *Classification* refers to the act of dividing a set of *entities* into *classes,* according to similarities and dissimilarities they bear to each other. The relation of an entity to a class is *is-instance-of* relation;

the entity is said to be an *instance* of the class. In object-oriented programming, the same situation is seen from a different perspective. The classes are first defined and the instances are then created according to the class definitions. The act of creating an instance is thus called *instantiation* (Stefik and Bobrow 1986).

- *Generalisation* refers to the creation of *generalisation taxonomy*, in which classes are related by *is-a* relation, between classes. The opposite of generalisation is called *specialisation* (Abiteboul and Hull 1987). Generalisations of a class are *superclasses* and specialisations *subclasses*. The resulting taxonomy is also called a *generalisation hierarchy* or *generalisation lattice* depending on its structure. The term *lattice* means, in this context, a directed graph without cycles (Stefik and Bobrow 1986). The terms *class hierarchy, type hierarchy, inheritance hierarchy* and *is-a hierarchy* (or lattice) are also used.[1]
- *Aggregation* is an abstraction for combining wholes from parts via *has-part* or *part-of* relation, which are inverses of each other. For example, a product is an aggregation of its components.

The dynamic ontology concerns the modelling of temporal aspects of the universe of discourse. There are various methods for modelling the evolution related to product families including different versioning mechanisms, schema evolution of databases, temporal databases and design modelling. The abstraction mechanisms and temporal aspects of conceptual modelling will be discussed in more detail in the following sections.

2.2 Is-instance-of Relation

The is-instance-of relation relates *individuals* (i.e., *instances, objects* or *data* as they are also called) to the *schema* (i.e., *classes, types, frames,* or *concepts*). The relation is typically established by classifying existing individuals or by instantiating class definitions.

In object-oriented modelling, individuals are called objects, which are a uniform programming element for computing and saving a state (Stefik and Bobrow 1986). An essential characteristic of an object is *object*

---

[1] In the original papers annexed to this thesis, the term *classification hierarchy* was used to refer to a generalisation hierarchy. In order to avoid confusion, in this summary part of the thesis, the term *classification hierarchy* is avoided.

identity, which allows referencing to the object. It is required that the object identity remains invariant across all possible changes to the object's value (Zdonik and Maier 1990).

The relation of individuals to the schema has different requirements or freedoms in different approaches.

- In databases and *class-based programming languages,* individuals are always assumed to have the structure described by the schema (Wegner 1990). For example, an individual may only have attribute values for attributes specified in the schema. Individuals are thus always required to *conform* to the schema.
- In description logic, the subsumption mechanism automatically classifies an individual according to its properties. Consequently, a change in the properties of an individual may result in the individual changing its class. In design, it is also useful to have some properties as classifying properties while the others are inherited after the design object becomes classified (Hakim 1993, Hakim and Garrett 1997).
- One approach is to separately record the is-instance-of relation between an individual and its class and then investigate the validity of the individual with respect to its class (Soininen et al. 1998).

In this thesis, the relation between individuals and the schema is investigated from the perspective of products. Study III addressed this issue and proposed that for products, the last view of the above ones is the most appropriate, but that will be discussed in due time later in the thesis.

The classification is not only applicable between the schema and individuals. In general, classification can form multiple levels, each level consisting of instances of the level above. Classes in the schema can be instances of meta-classes. At some point, however, the creation of new meta levels is terminated. Typically, the termination is made at a special level with a class being an instance of itself (Stefik and Bobrow 1986, Peters and Özsu 1993, Mylopoulos 1998).

2.3 Is-a Relation

Classes are traditionally organised into taxonomies by is-a relation in the sense that "Student is-a Person". Person is a *superclass* of Student and

Student is a *subclass* of Person or, in other terms, "Person is a generalisation of Student" and "Student is a specialisation of Person".

There are two main views on semantics of is-a relation and how to establish the relation between classes.

In a cognitive view, classes and a class structure serve as a tool for storing facts in our minds. One guideline for creation of a generalisation hierarchy is thus *cognitive economy,* that is, to provide "maximum information with the least cognitive effort" (Parsons and Wand 1997). This kind of approach, which starts creating the concepts from individuals (i.e., objects), is called an *object-centred view* (Goyal et al. 1992). The *subsumption* of description logic works with the same principle, by organising the existing concepts into a generalisation hierarchy according to their properties. Subsumption means that anything which is true of the general concept must also be true of the more specific concept derived from it (see, e.g., Searls and Norton 1990, McGuinness and Wright 1998a).

In a *class-centred view,* e.g., in object-oriented programming, the class hierarchy is defined for a certain purpose (Goyal et al. 1992). The hierarchy is assumed rather static and classes are typically not organised automatically. In object-oriented programming, one particular property of generalisation is *subtyping*. It means that instances of a subclass can be used where an instance of the superclass is expected (Nierstrasz 1989, Zdonik and Maier 1990).

Inheritance is a property in an object-oriented language that allows defining classes based on the definition of other classes. The properties of a class are inherited by its subclasses. A class can then define its own properties in addition to the inherited properties and so distinguish itself from its superclass. According to the model, it may be possible to refine the properties in subclasses or even entirely *override* (or *substitute*) them (Stefik and Bobrow 1986).

In *uniform object* approaches in which there is no distinction between classes and objects, the *delegation* mechanism implements the sharing of properties between objects. Each object independently delegates its properties to other objects (Lieberman 1986, Almarode 1989).

Is-a induces a directed graph, which is required to be acyclic and is thus called a *class lattice* or *inheritance lattice* (Stefik and Bobrow 1986). This means that a class may have multiple superclasses but not even indirectly be a superclass of itself. In the case of multiple superclasses, the class inherits all properties of those classes, and therefore, this case is also called *multiple inheritance.* Multiple inheritance creates multiple inheritance

paths for a class, which may lead to difficulties, for example, if the properties are identified by their names, and properties with the same name are inherited multiple times. Because of the difficulties, multiple inheritance is not always supported and is-a is required to induce a tree (or trees).

For a particular universe of discourse, multiple different ways exist for defining the concepts and organising them into taxonomy (Parsons and Wand 1997); in other words, design of a generalisation hierarchy typically involves trade-offs (Taivalsaari 1997). Consequently, if a generalisation hierarchy, i.e., especially in the case of single inheritance, is constructed mainly for the purposes of inheriting common properties, the result may not be optimal from the conceptual viewpoint. In this thesis, the generalisation is primarily considered as a conceptual tool for organising classes and secondarily as a mechanism for inheriting properties. In other words, the generalisation would be constructed from the viewpoint of product families, not with optimal inheritance in mind.

Generalisation is a powerful data modelling mechanism that can be naturally applied to industrial products (Jørgensen 1994). Component data, in particular standard components that are purchased, must be organised for many reasons, such as efficient searching of components for new designs and various groupings for statistical analyses of component purchases, stock value, wear-out in use, etc. The PLIB standardisation effort addresses generalisation for component libraries, especially taking into consideration parametric components (ISO 1995b, ISO 1996).

For industries in which after-sales is of great importance it is beneficial to refer to the parts of delivered products with a standard terminology. With such a dictionary on top of a database, one can, for example, find all delivered products that use a particular kind of component. The required type information of subsystems and components can be organised into a class hierarchy or hierarchies. Within STEP, this kind of work has been carried out in the development of AP 221 for process industries under the name STEPlib (ISO 1997a, ISO 1997b, Owen 1997).

2.4 Has-Part Relation

The part-of relation is studied from the philosophical point of view in *classical extensional mereology (CEM)*. The word *mereology* is derived from the Greek work *meros* meaning *part*. The basic idea is that a whole consists of

parts so that the whole is actually defined by its parts. Simons (1987) has given an axiomatisation for the part-of relation, which consists of the following axioms (see e.g., Gerstl and Pribbenow 1995):

- EXISTS: If A is part of B, both A and B exist.
- ASYMM: If A is part of B, B is not part of A.
- SUPPL: If A is part of B, there exists such C that is part of B and there is no X which is both part of A and part of C (i.e., B also has part C that is disjoint from A).
- TRANS: If A is part of B and B is part of D, A is part of D.
- EXT: "wholes with same parts are identical."
- SUM: "There is a unique mereological sum S for any non-empty set of existing individuals."

There are some controversial issues with respect to how people understand part-of relationship (Gerstl and Pribbenow 1995). Identity, for example, is in classical mereology tied to the parts that make the whole. In many cases, the identity is considered independent of changes of parts. For example, changing the tyres of a car is not perceived as resulting in a new car. However, after changing the shaft and the head of a hammer it may be difficult to speak of the same old hammer. In the strict object-oriented sense, however, the identity of an object is its immutable property independent of its other properties. This means that changing of any or all parts of an object has no effect on its identity. If one feels that after replacing all parts of an object, it is not the same object any more, one must create a new object with a new identity.

The intuitiveness of the SUM axiom can also be criticised since it says that there exists a whole for any set of objects; most of such "randomly" defined wholes are meaningless (Törmä 1997).

The transitivity axiom TRANS is valid in a certain sense within the physical artefacts, but is already controversial when the part-of relation is extended, for example, to member relation (Winston et al. 1987, Artale et al. 1996). For instance, is an arm of a violin player a part of the orchestra?

Classical mereology concerns the part-of relation between individuals only. The above axioms, therefore, are not valid for the schema level representation of part-of relation. For example, an individual cannot be a part of itself since that contradicts the ASYMM axiom. In the schema, however, it is meaningful to make the general statement "component has-part component"—two individuals, both of the type component can

be in part-of relation to each other if they are distinct individuals. Classical mereology thus forms a philosophical foundation for has-part relation, especially in the sense of modelling the real world around us, but as such it provides little for the practical product family modelling tasks.

Halper (1992) has addressed the part-of relationship in practical terms and has discussed its semantics on the schema level, i.e., not only as a relationship between individuals. Halper identifies the following kinds of part-of relationship:

- *Exclusive* and *shared*. Part-of relationships are divided by whether or not a single component individual can be simultaneously an immediate part of multiple wholes, e.g., whether a motor individual can be a part in multiple car individuals. There are three subcases: *shared part relationship*, *class exclusive part relationship* and *(global) exclusive part relationship*. These specify whether a component individual can be shared, can only be shared by instances of different classes or cannot be shared at all, respectively. The class exclusiveness captures requirements, such as a scientific article can be a part of a journal and a compilation, but not a part of two journals or two compilations.
- Another distinction is made between *single-* and *multi-valued part relationships*. For example, a car having 2 to 5 doors is a multi-valued part relationship. These have subcases: *single-valued, multi-valued, range-restricted, fixed-cardinality, essential* and *multi-valued essential*. These cases are not mutually exclusive; for example, essential part relationships is a fixed-cardinality relationship with multiplicity 1.
- As an example of *ordered part relationships,* Halper (1992) gives memo, which has multiple text blocks as parts that can be enumerated and have an order, e.g., header, body and trailer. This is an *ordered part relationship of definite number*. Alternatively, the chapters of a book are modelled using *ordered part relationship of indefinite number*.
- *Dependency* is related to the deletion. For example, when a whole is deleted, its parts may also be deleted automatically. A part relationship can be *part-to-whole dependent* or *whole-to-part dependent*.
- Values of certain properties can be propagated via part relationships. For example, the age of a car may be determined by the

> age of its frame and the colour of the doors may be the colour of the body. The former is an *upward propagating part relationship* while the latter is a *downward propagating part relationship* (with respect to the particular property values).

Halper (1992) then discusses the combination of all these aspects and derives the conclusion that including them all allows inconsistent schemas.

Artale et al. (1996) had made a survey on the concepts for modelling part-of relations. The main thesis of their paper is that part-whole relations should not be modelled as ordinary relations and the burden of capturing their special semantics left to the designer. They make various distinctions in kinds of part relations. For example, they make a distinction between *implicit* and *explicit* modelling of wholes. That is, values of the attributes wife and husband in objects representing humans define implicitly wholes that can be called families. Alternatively, a family may be represented explicitly by a separate family object, which has two parts, wife and husband. In addition, they distinguish between *vertical* and *horizontal relations*. Vertical relations model how the existence or properties of the whole depend on the existence or properties of its parts, or vice versa. Horizontal relations describe the integrity of the whole as constraints between the parts.

One object-oriented approach for modelling part-whole relations is that of ORION (Kim et al. 1987, Kim et al. 1989). It has *composite objects,* for which the part relation is defined by means of *composite reference.* For example, the class Vehicle may have a composite reference through *composite attribute* Body to the class Autobody, which acts as the domain of the composite reference. The subclasses of Vehicle inherit the composite attribute. The composite attribute corresponds to the term *part name* of some others (Artale et al. 1996, Soininen et al. 1998).

Mereology focuses on the philosophical and conceptual basis of the part-whole relation of single entities. Object-oriented modelling of composite objects provides an intentional view to the has-part relation. With product families, these do not suffice. Product families include multiple structural variants and thus their representations need to incorporate some generic aspects for modelling that variety. The next section introduces different means for modelling product variety.

2.5 Product Variety

Product variation has become a common phenomenon. Most companies now offer large product variety in order to compete in marketplace (Sanderson and Uzumeri 1997, Ho and Tang 1998). From product data modelling viewpoint, two major types of variation are distinguished here: *variety within components* and *structural variety*. The variants of a component coexist representing the alternative designs of the component. A similar form of variety is provided by parametric components (ISO 1995b, ISO 1997c). That is, a component has parameters, typically some dimensions, which can be given values. These both model the *variety within a component*. A different form of product variety is *structural variety*, which refers to the variation in the product structure. For example, for a personal computer there may be a selection of different keyboards as *alternative parts,* and an external microphone as an *optional part*. Alternative and optional parts are typical forms of structural variety. The relation of the terms *variant* and *version* will be addressed in the next section.

This thesis concentrates on the structural variety of products. It is sometimes a matter of judgement whether the variety is modelled within a component or as structural variety. For example, various hard disks for a computer may be represented by a component type with a parameter for its size or all disks of different sizes may be separate component types. The choice depends on the intended usage of the representation, as in all modelling.

Product configuration is an active area of research in which the representation of product variety is central. The focus in this thesis is on the modelling of product families. Therefore, the approaches to product configuration are only described from that perspective. For a more thorough view on product configuration, the reader is referred to the special issues published recently (Faltings and Freuder 1998, Darr et al. 1998) and the reports from the AAAI workshops on configuration (Faltings and Freuder 1996, Faltings et al. 1999).

There are several ways of modelling product variety. Some methods are based on explicit representation of the product structure, some represent the possible and valid sets of components and not necessarily the product structure at all (Männistö et al. 1996). The latter kind of methods are especially suitable for products in which the basic architecture is well known and a product individual can easily be assembled from the given set of components. Instead of or in addition to a product structure, a method may model the connection of components into a

network of some sort. Some methods enable automatic search for solutions to the configuration task after some initial selections are made. Naturally, various combinations of basic methods are used. In the following, methods based on product structure description are introduced first.

- *Set of BOMs* is a trivial case and not actually a product family model. The method, however, is widely used in the industry, even for modelling large numbers of variants.
- *Generic, variant and parametric BOMs.* These define alternatives for certain components in a BOM and possibly global variables or selection tables for determining the choice. (van Veen 1991, Schönsleben and Oldenkott 1992, Tiihonen et al. 1995, Erens 1996)
- *AND-OR graphs,* in which each node either breaks down to parts (AND) or to a selection between choices (OR). The AND and OR levels alternate, so that an AND level is followed by an OR level and vice versa. (Kemper and Moerkotte 1994)
- *Combined class 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 in its subclass (a kind of an OR). This approach is used in the PLAKON model (Cunis et al. 1989) and in SAP Configurator (SAP 1994).
- *Classless object models.* One potential approach for modelling product families is the uniform object model. The approach abandons the distinction between classes and instances and models everything uniformly as objects that can inherit (by delegation) properties from each other (Lieberman 1986, Stein 1987). The approach has been used for design modelling (Zucker and Demaid 1992b) and for product family modelling (II).
- *Composite Instance Variable.* Composite instance variables are used in object-oriented models for representing components (Kim et al. 1987, Kim et al. 1989). Such a variable actually represents a set of components if its domain is a class that has subclasses; any of its subclass being an eligible component type.

Other methods that have been used for describing the valid sets or combinations of components include:

- *If-then rules*. In many expert systems, if-then rules can be used for implicitly describing product structure (Hayes-Roth 1985). For example, *IF component A OR B is in configuration THEN component X must be included, too*.
- *Description logic* provides a powerful conceptual modelling mechanism with deduction capabilities, which include a subsumption for automatic classification. Description logic has also been used for configuration modelling and solving the configuration task (Weida 1996, McGuinness and Wright 1998b).
- *Compatibility and incompatibility constraints* state which components can or cannot be together in a configuration, respectively. These constraints can be recorded as n-ary tuples and various algorithms exist for satisfying the constraints (Mittal and Frayman 1989, Mittal and Araya 1992, Sabin and Freuder 1996, Weigel and Faltings 1996, Fleishandler et al. 1998, Soininen and Gelle 1999).
- *Interfaces, ports and connectors* provide deeper modelling for compatibility of components. The basic idea is that certain components need, can or cannot be connected. This is modelled by ports and connectors that must fit together in a valid configuration (Mittal and Frayman 1989, Tiihonen 1994).
- *Resources*. In resource-based modelling the idea is that some components *consume* something, such as power, fuel, ventilation, etc., while other components *supply* these resources. In a valid configuration, all resources need to be in balance (Heinrich and Jüngst 1991, Heinrich and Jüngst 1996, Jüngst and Heinrich 1998). For example, the total produced net power must be more than maximum consumption.

Regardless of research and commercial activities, there is still no universal model for product configuration modelling. In addition, the long-term maintenance of configuration knowledge has been known to be a major problem for some time. For example, one of the earliest examples is the XCON configurator, which after a few years of operation was reported to provide work for tens of developers and maintainers (Barker and O'Connor 1989). However, there is still amazingly little work on the evolution aspects of product configuration, besides this thesis. The problem has mainly been referred to as a promising area for future research.

2.6 Versioning

Version is a concept for describing the evolution of products (Katz 1990). In addition, versions can be used for various other purposes, such as concurrency control, recovery, performance enhancement and update-free databases (Dittrich and Lorie 1988). Therefore, no single semantics for versioning exists. In the following, some of the most common ones are described (Björnerstedt and Hultén 1989, Katz 1990, Feiler 1991, van den Hamer and Lepoeter 1996).

- *History.* Versions can represent historical development of designs. In other words, a version is a semantically meaningful snapshot of a design object at a point in time. Version history allows one to go back to previous versions.
- *Experimental path of development.* In a design process, multiple versions may be generated as alternative solutions with the idea that one of them is finally selected for further development.
- *Variants.* One product can have two slightly different variants, for example, for different market areas. The variants are intended to co-exist.
- *Concurrent development.* Separate versions can be used for enabling concurrent modifications of a design. The separate versions are subsequently merged into a single one.
- *Changing types.* Object versioning may be needed to cope with changing type definitions so that the instances of the old type can be retained with the old type version.

Many of other terms, including *revision, equivalent representation, alternative* and *variant*, have been introduced for describing versioning and related phenomena (Katz 1990).

The concept of a version can represent the evolution of many kinds of objects, such as source files, product components or design objects in a very general sense. Usually, no attempt is made to define the precise meaning of a version, although some authors have required that all versions of an object are implementations of the same interface (Batory and Kim 1985, Joseph et al. 1991). In industry, a new version is typically required to be substitutable, i.e., it can be used in place of older versions. This is sometimes expressed by saying that versions are compatible with respect to "form, fit and function" (van den Hamer and Lepoeter 1996).

Many authors relate a versionable object with a set of versions. Such an object with a set of versions is called, for example, a *generic version*, *generic instance* or *generic object* (Katz et al. 1986, Kim et al. 1987, Kim et al. 1989, Biliris 1989). Typically, a generic object works as a point of reference for the set of its versions and can contain the definitions that are common to all of the versions in the version set. A set of versions, as such, is a rather general notion for modelling versioning, and therefore, many approaches propose structure between versions. Dittrich and Lorie (1988) use clusters to organise the version set of a design object. They show an example where *design object* has *alternatives*, which have *revisions,* which in turn have *versions*. The top left-hand corner of Figure 2 illustrates such an organisation. Kemper and Moerkotte (1994) give a similar hierarchy, but have the concepts in the order *object, representation, alternative* and *version,* which is essentially the model in DAMASCUS system (Mülle et al. 1988). This is illustrated in the top right-hand corner of the figure. Kemper and Moerkotte also give examples of more complex hierarchies with *alternatives, representations*, again *alternatives*, and then *versions*. Some companies have products that have *revisions,* which in turn have *variants*, in that order (Peltonen 2000). This is shown in the bottom left-hand corner of Figure 2.

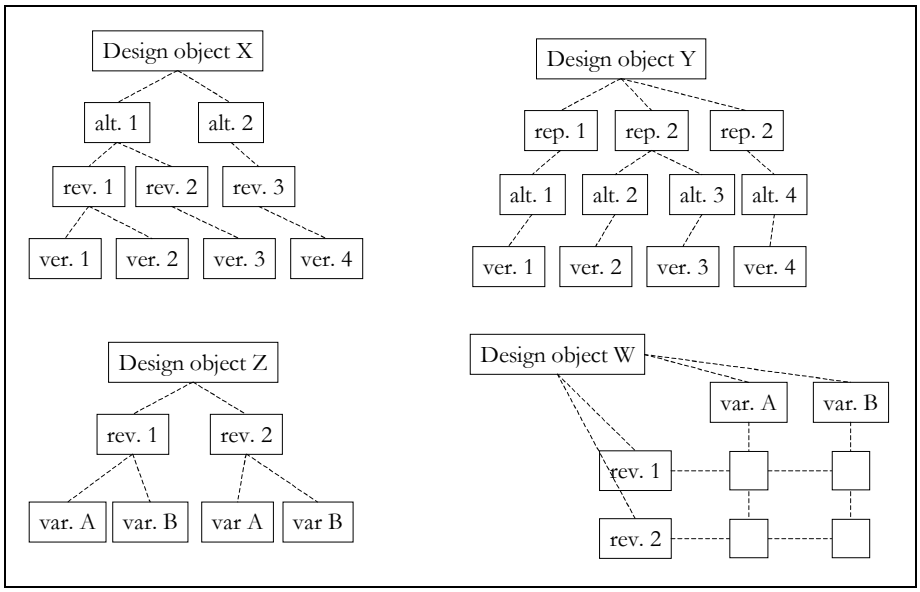

Figure 2. Different hierarchies for clustering versions of a design object.

It has also been argued that the terms *revision* and *variant* cannot be ordered in such a fashion, because they are actually orthogonal to each other (Reichenberger 1989, Wedekind 1994). The orthogonal dimensions identified by Estublier and Casallas (1995) are *temporal, logical* and *cooperative*, which correspond to the terms *history, variant* and *concurrent development* of the list above. Assume that in the case of the bottom left-hand corner of Figure 2, variant 'var. A' has the same intended meaning in both revisions 'rev. 1' and 'rev. 2'. Thus, the concepts *revision* and *variant* are actually orthogonal and should be represented as in the bottom right-hand corner of Figure 2, where each empty box would store the data related to a particular revision and a particular variant of the design object.

There can be two kinds of *component references* to versions: *statically bound* and *dynamically bound* (Kim et al. 1987). A statically bound reference specifies explicitly a component and one of its versions. Allowing only statically bound references has severe limitations, for example, rebinding of references must be considered each time when a new version is introduced. It is, therefore, common that some dynamic version referencing mechanism is introduced.

A dynamically bound reference is a reference to a generic object and is called a *generic version reference*. A generic reference is intended to be bound to "one version from the set" (Chou and Kim 1986). A resolution mechanism is needed for telling which version from the version set is the correct one in a particular usage of a generic version reference. A generic reference can also be bound at various points in time.

The idea of a *current version* is that at any time one of the versions, i.e., the current version, is a valid representative for the generic object. A generic reference is bound to the version that is the current version at the resolution time. The current version can be explicitly stated or the resolution may be based on time. The latter is appropriate when versions represent the evolution in time; in resolution of variants, other mechanisms are needed.

In the context of products, the versioned objects form part-of structures. Therefore, one particular issue that needs to be addressed is the semantics and effects of a change in one component with respect to the entire part-of structure; a phenomenon also called *change propagation* (Katz 1990). Sometimes the term *change propagation* means propagation of a change in the schema to the instances of that schema (Nguyen and Rieu 1989b, Casais 1990).

Change propagation deals with the question of whether the creation of a new version of a component leads to the creation of new versions of assemblies that contain the component. If new versions are created automatically and recursively all the way up to the root of the product structure, the result is a large number of possibly unnecessary versions (Skarra and Zdonik 1986, Katz and Chang 1987, Katz 1990, Ramakrishnan and Ram 1996). Version substitutability is one of the most practical ways to limit change propagation. That is, if the new version is form, fit and function compatible with the old one, there is no need to propagate the change (van den Hamer and Lepoeter 1996). Complete change propagation was close to impossible with manually maintained parts lists. Modern product data management systems enable easy propagation of changes in product structures, even when this creates a large number of new versions. However, it still makes sense to stop the propagation at the appropriate level (van den Hamer and Lepoeter 1996).

A version is typically related to its predecessor by *is-derived-from* relation (Katz 1990), sometimes also called *is-a-derivative-of* (Wagner and Lima 1991). It is possible to distinguish between *successor* and *is-derived-from* relations. Is-derived-from denotes the origin of the new version while successor represents the concept that the new version replaces the current version in use. In many cases, the successor of an old version is also derived from the old version, but this need not always be the case.

In a design process, versioning has some special semantics. In addition to deriving new revisions and variants, designers experiment with alternative solutions. These alternatives are typically represented as new versions derived from the current version. At some stage, experiments are over and one of the alternatives is selected as the next current version (Ramakrishnan and Ram 1996). A slightly more general approach is to allow multiple version derivation hierarchies within a version set (Ahmed and Navathe 1991).

Most of the approaches to versioning in modelling the evolution of product designs reflect the usage of versions as intermediate stages of a design object during a design process, and alternatives typically represent tentative development lines. As pointed out by Biliris (1989), in design it is more important to keep track of the current status of design objects than preserve historical information.

2.7 Schema Evolution

In this section, research on schema evolution of databases is first presented in its pure form and the relation to product family modelling is drawn only thereafter.

Each database has a data model, also called a schema, typically defined using a data definition language (Ullman 1988). Schema evolution has been an active area of database research addressing the problems that arise from the changes to the schema of a database (Roddick 1992, Roddick 1995). The most central problem is what to do with the existing population of data instances when the schema is changed. In addition, the new data should also be viewable though old schema definitions (Roddick 1995). The approaches to schema modification, especially if the schema contains classes, can be classified to (Casais 1990, Gibbs et al. 1990):

- *Tailoring,* in which the new aspects are achieved by deriving new subclasses.
- *Surgery* is based on a certain well-defined set of primitive operations, e.g., 'add method' or 'change class name', that can be combined to modify classes. An example of class surgery operations is the taxonomy of schema changes given by Banerjee et al. (Banerjee et al. 1987, Banerjee et al. 1987).
- *Versioning* enables the recording of the history of class modifications. In class versioning, the existing class definition is not changed but rather a new version of the class is created (Monk and Sommerville 1993).
- *Reorganisation* of a class library is needed when major changes are made to the classes and relationships between them.

To clarify the terminology of the research dealing with schema evolution, Roddick (1995) gives the following definitions for the basic terms. The definitions are based on the concept of data population and reflect the database researchers' view of schema evolution.

- *Schema modification.* A database system allows changes to the schema definition of a populated database.
- *Schema evolution.* A database system facilitates the modification of the schema without a loss of existing data.
- *Schema versioning.* A database system allows accessing of all data, both retrospectively and prospectively, through user definable

version interfaces. Schema versioning is *partial* if only viewing is supported both retrospectively and prospectively and *full* if updates are supported too (Roddick 1996).

After a schema modification, the data population needs to reflect the changes in the schema (Nguyen and Rieu 1989a, Tanaka et al. 1989). The approaches to change propagation can be classified to (Penney and Stein 1987, Casais 1990):

- In *filtering*, or *screening* (Banerjee et al. 1987), database manager needs to provide filters to allow an operation defined for a particular *type version* to be applied to an instance of any version of the type (Skarra and Zdonik 1986, Skarra and Zdonik 1988).
- In *conversion,* the modified instances are stored in the database. The conversion can be *eager,* in which case all instances are converted right after the schema modification, or it can be *lazy* (Tan and Katayama 1989), in which case the conversion takes place only when the instance is accessed. Lazy conversion is also called *incremental conversion* (Björnerstedt and Hultén 1989).
- *Change avoidance* is an approach in which only changes that do not necessitate modification of instances are made, for example, adding a new class or renaming an attribute (Casais 1990).

In addition to changes to data, i.e., *structural changes*, the consequences of a change to the programs accessing the data, i.e., *behavioural changes,* have been considered (Skarra and Zdonik 1986, Skarra and Zdonik 1988, Zicari 1991, Clamen 1992, Zicari 1992, Peters 1994). A *behavioural consistency* means, for example, that a method of a class does not result in an unexpected error after a schema change.

A different approach to schema evolution is provided by description logic with the idea not to explicitly define the generalisation hierarchy but have it implied by the properties of the classes. In a model defined by Peters and Özsu (1997), the class lattice is formed based on *essential properties* and *essential supertypes* (i.e., superclasses) of a class, which are explicitly stated for each class. Hakim defined *sufficient conditions* that determine whether an instance is an instance of class (Hakim 1993). In this approach, the focus is on the design process, which is supported as instance evolutions. The approach exhibits a different kind of schema evolution because, if the sufficient conditions of a type are changed, then, instead of being converted, its instances may be reclassified and thus no longer be its instances.

Monk (1993) has pointed out that there is more to a change than the structural conversion. For example, changing the attribute 'name' for a class 'person' to 'surname', is more than just renaming the attribute, which is an operation that does not necessitate conversion of instances. The old attribute 'name' could have stored values such as "Teppo Ilkka", "Jones, T. A." and "Otto von Bismarck". Converting all possible values of 'name' to surnames only is thus not trivial. Skarra and Zdonik (1986, 1988) also discuss the semantics of class versions, i.e., what is needed for two definitions to be versions of a class instead of being two classes. They do not provide any solution; in fact their model allows "a conversion of a frog to a prince", as they put it. They, however, do assume that most modifications will involve only minor alterations, which their model is actually meant to support.

The previous research in database schema evolution is next compared with the problem area of the evolution of configuration models (see Männistö et al. 1996). Product families are typically modelled using classes, and therefore, modifications of them are more schema than instance level operations. For example, for a company that manufactures a large variety of products, the addition of a new product variant is more a schema operation than a modification of an instance. Similarly, the company needs to retain the history of the schema to efficiently support its after sales processes.

The assumption in database schema evolution is that instances can be converted. This assumption is based on the principle that the schema constantly represents the same (or at least almost the same) world and a schema modification only alters the way the entities are represented. Therefore, the conversion of old instances to the new schema is meaningful. A product family description, however, represents product individuals to be sold and manufactured. When the description is modified, it does not represent the same set of product instances from a different angle; it represents an entirely new set of product individuals. The old product individuals are not represented by the new description because they contain old components, for example. Thus, a conversion of the representations of old product individuals to the new schema is often not meaningful. This shows the different role of the schema in databases and in product family modelling. This distinction is a major differentiating factor between the schema evolution research in databases and this thesis.

2.8 Evolution of Individuals

In addition to the evolution of schema, of importance also is the evolution of individuals. In many systems, the data is just modified without keeping the history of changes. When needed, the temporal aspects are sometimes modelled, for example, by means of DATE type attributes. This, however, is not an adequate solution as the system gives no support for the semantics of manipulation of temporal aspects (Snodgrass 2000). Temporal databases provide a solution for capturing the time with data. There are, however, also other approaches for modelling the evolution of individuals, of which some are summarised as follows:

- *State change according to the schema* refers to modification of data, i.e., change of attribute values, in a database. When time is recorded for each change, the result is a temporal database (Goyal et al. 1992). There are three typical *temporal events* that cause objects to evolve: *update, delete* and *insert* (Su et al. 1998).
- *Object migration* is needed, for example, to support an employee becoming a manager. That is, an individual changes its class, which is also called *dynamic multityping* (Goyal et al. 1992).
- *State change independently of the schema.* Evolution of object may also be independent of the schema, so that properties may be given to the object and its membership in classes is investigated thereafter. This kind of evolution with dynamic classification is useful for supporting design processes (Hakim and Garrett 1997).
- *Individual evolution without the schema.* In uniform object approaches, there is no schema, only individuals (or objects, as they are called). Each individual describes its own properties. An individual may be used as a prototype in the creation of new individuals, which captures the evolution of individuals similarly to the way in which prototyping does in design (Demaid and Zucker 1992, Zucker and Demaid 1992a, Zucker and Demaid 1992b).
- *Object evolution beyond the schema.* In after-sales processes of industrial companies, product individuals may be modified by customers practically as they please (Männistö et al. 1996). In this kind of evolution, the individual is not entirely represented by the schema but may still have a representation understandable to an intelligent observer.

The support for temporal aspects may need to consider the time of change. Changes may be recorded at a time different to the time of the change. This has led to the distinction of most important times of change: *transaction time* and *valid time* (Goyal et al. 1992, De Castro et al. 1997, Snodgrass 2000). The former refers to the time the change is recorded in a database whereas the latter to the effective time of the change. These times are considered orthogonal (Jensen and Snodgrass 1996).

A major issue in conceptual modelling with time is the nature of time and what properties it has (Theodoulidis and Loucopoulos 1991). Decisions should be made in the following ontological assumptions about the nature of time (Theodoulidis and Loucopoulos 1991, Jensen and Snodgrass 1996, Bettini et al. 1998, Goralwalla et al. 1998, Jensen et al. 1998):

- Whether valid time or transaction time or both are supported.
- Whether *time primitives* are *anchored* (*absolute*) or *un-anchored* (*relative*), e.g., X happened last week. There are two kinds of anchored time primitives: the *instant* (*moment, chronon*) and the *interval*. An interval is the duration between two specific instants.
- Whether time is *discrete,* i.e., time primitives are isomorphic to integers, or *dense,* i.e., time primitives are isomorphic to real numbers.
- Whether time is *determinate* or *indeterminate*. The distinction is whether an event occurs "during the whole time instant" or sometime within the instant. For example, "a person is in a train the whole day" or "a person leaves Paris that day", respectively.
- Whether time is *linear, non-linear,* e.g., *branching* to different future scenarios, or even *periodic,* such as weekly meetings.
- Whether time has *variable span,* e.g., length of a month, which varies between months, or *fixed span,* e.g., one hour.

There is no need for retrospective updates in design databases, or more accurately, the versioning mechanism is also set up to handle corrections to old designs. Therefore, in design databases transaction time and valid time can be treated as identical (Biliris 1989). The temporal aspects needed in this thesis are thus valid time with anchored primitives with determinate discrete instants and determinate discrete intervals.

3 Terminology

In this section, a brief summary of the basic terms used in this thesis is given. This is not a complete terminology. The purpose of this section is to familiarise the reader with the terminology of this thesis and to relate that to the reader's own, possibly different, terms for the same concepts.

This work concerns *product data modelling*. The term *model* means sometimes a modelling method with its basic concepts, e.g., ER-model, object-oriented data model or relational model. Sometimes, however, a schema level description is called a *model,* e.g., *configuration model*. In addition, a *product model* can also refer to a description of a single product, which is typical usage for large products such as ships or power plants. In this thesis, the term *model* is used in the both meanings mentioned above with the assumption that the context clarifies the meaning. To avoid confusion, the term *product model* is avoided.

In common language, the term *product* is used both on the type level and on the individual level. For example, a representative of a company may well say "our best-selling *product* is ZaXi" and "if our *product* breaks down, we'll replace it within 48 hours". The former refers to *product types* while the latter refers to *physical products,* also called *product instances* or *product individuals*. In normal communication, the distinction between types and individuals is typically clear from the context or not of great importance. The term *type* will be used as a synonym to *class,* also used in object-oriented modelling, to *entity type* of E-R modelling and to the terms *frame* and *concept* of other approaches. A collection of types forms a *schema*. The term *individual* is a synonym for the term *instance*. Sometimes the set of individuals is also called *data* or *extension* (in particular that of a class). The term *object* is mainly used in this thesis as a general data modelling concept that incorporates both types and individuals.

In general, it is impossible to distinguish between a *product* and a *component* because that depends on the viewpoint. What is a product to someone may be a component to someone else. Therefore, the terms are regarded as rough synonyms here. The term *product* is mostly used as a general term referring to the whole product and a *component* more as a technical term for the pieces of a product but it may also refer to the whole product. Components of a product do typically form structures, called simply *product structures*. Another closely related term is *part*. In this thesis, the term *part* is reserved for the relation between components that defines a product structure. That is, a component may have other

components as its parts; they are in *has-part* relation. Similarly, a component may be a *part-of* another component. Has-part and part-of describe the same relation but from different directions. In other contexts, the terms *assembly, aggregation, compound object, composite object* and *complex object* may refer to an object that has parts.

Data models are needed for products on both levels of abstraction, the type level as well as the individual level. A product structure describing a physical product is called a *specific product structure*. A *physical product* is manufactured according to a specific product structure. A specific product structure is sometimes called a *Bill-of-Materials* (BOM). This is in contrast to a *product family description*, which describes a number of different specific product structures. A product family description is considered here synonymous to the terms *configuration model, generic product structure* and *generic Bill-of-Materials (GBOM)*. Product family descriptions are needed when the number of *product variants* becomes too large to simply enumerate. Sometimes the terms *configurable product* and *product with a large number of variants* is used for referring to such products.

A *configuration process* is a process for producing a specific product structure from a configuration model for a specific customer need. In this case, the specific product structure is also called a *configuration*. However, sometimes also the configuration process is called *configuration*. In this thesis, the term *instantiation* also refers to creation of an instance.

The evolution of products is typically modelled by means of *versions*. Other similar terms, such as *alternative, revision* and *variant* are used. The term *alternative* refers typically to the alternative designs during a design process, which are not addressed in this thesis. The term *revision* refers to improvement of some sort with the intention of replacing the old version. The term *variant* refers to a version that is meant to exist in parallel with other variants. For example, a company may manufacture different variants of a product for different market areas. In the terminology of software configuration management (SCM), *version* is a general term with two main special cases: *revision* and *variant*. In many cases, the term *version* is also used in the sense of the term *revision* of the SCM terminology.

In this thesis, the focus is on the temporal aspects of versioning and structural variation (not the variation within components) so the distinction between the terms *revision* and *variant* is not needed. Therefore, the term *version* is used to refer to temporal evolution of objects.

4 Aims of the Study

The aim of this study was to develop a methodology for modelling product families and temporal aspects in them. The principal aims stemming from that were to understand the fundamentals of modelling product families and to understand the evolution of product families as a phenomenon as well as its implication to information technology, especially to the data representation mechanisms needed. These general aims were divided into more detailed ones as follows:

(1) To understand the problems in modelling product families with the concepts *is-a relation*, *has-part relation* and *version* and to explain why the problems differ from those addressed in traditional data modelling.

(2) To investigate the suitability of the STEP product data modelling standard, which is the largest integrated effort in the area of product data modelling, for modelling product families (I).

(3) To examine the classless object approach for modelling the configuration process of product families and for representing product families with uniform objects (II).

(4) To provide a conceptual framework for modelling the evolution of schema and individuals of a product family with concepts *is-a relation* and *has-part relation* (III).

(5) To investigate the reconfiguration of product individuals in Finnish industry and propose conceptual framework for representing their reconfiguration knowledge (IV).

(6) To develop a conceptual model for representing individuals of product families on the basis of experiences with an industrial company in situations where the individuals are different and the level of detail by which the individuals are known varies (V).

5 Results

5.1 Modelling Product Families

The problem area of this thesis is product families and evolution related to them. The area was approached from the conceptual modelling perspective. This work, therefore, aims at finding concepts and semantics of product family and individual evolution. More concretely, the goal is to describe a basis for an advanced PDM system that enables the support of product family management policies of a company, in addition to basic concepts, such as versions of various entities. It is also important to control the effects of versioning, for example, in product structures and in generalisation hierarchy. Currently, such policies are mainly controlled by company standards or traditions. It would be beneficial to encode these policies in an advanced PDM system, which would then not only control them, but also help in conveying the policies, for example, to newly acquired organisations.

Certain special aspects of product family modelling characterise their evolution. Therefore, the modelling of product families, especially the structural variation, is next addressed by investigating the problems in modelling product families within the STEP standard (I).

5.1.1 STEP and Product Families

STEP, which is officially known as the ISO-10303 standard, is a large undertaking for standardising the modelling of industrial products for their complete life cycles (ISO 1994a, Owen 1997). STEP is based on schema-individual paradigm with a static schema, which in STEP becomes standardised. Therefore, STEP provides an excellent case for investigating how product families can be modelled on top of a strict schema-individual paradigm. Study I of this thesis conducted a survey to STEP in the area of product family modelling; results of which are summarised in this section.

The idea in STEP is to provide means for defining application specific concepts for modelling products in a particular application area. The application specific concepts are standardised into parts of STEP called Application Protocols. STEP includes the Application Protocol 214 (AP214), which, among other things, addresses the representa-

tion of product families. In Study I, the First and Second Committee Drafts of AP214 (ISO 1995a, ISO 1997d) were examined in addition to the basic constructs of STEP (ISO 1994a, ISO 1994b, ISO 1994c, ISO 1994d, ISO 1994e). It should be noted that this thesis is only concerned with certain aspects of product structure modelling, clearly excluding, for example, geometry, which constitutes a major part of STEP.

STEP also includes special parts, such as EXPRESS-X language (ISO 1999), which is a mapping language for converting individuals of one schema to individuals of another schema. EXPRESS-X helps in schema evolution and can also be used for mapping data from a company specific schema to a standard schema.

Study I argued that the approach proposed in AP214 for modelling product families is not adequate. Moreover, it was identified that the reasons for the inadequacy were not due to the draft status of the Application Protocol, but are more fundamental in nature. These reasons will be explained next, but before going into the problem, the structure of STEP is briefly reviewed.

Figure 3 shows the overall relation of the main parts of STEP. One of them is EXPRESS (ISO 1994b), which is a modelling language for defining the other parts of STEP including Integrated Resources and Application Protocols. These parts defined using EXPRESS become parts of the standard, which means that they are characteristically static, corresponding to a database schema. Companies would then describe their product families as instances of the standardised schema.

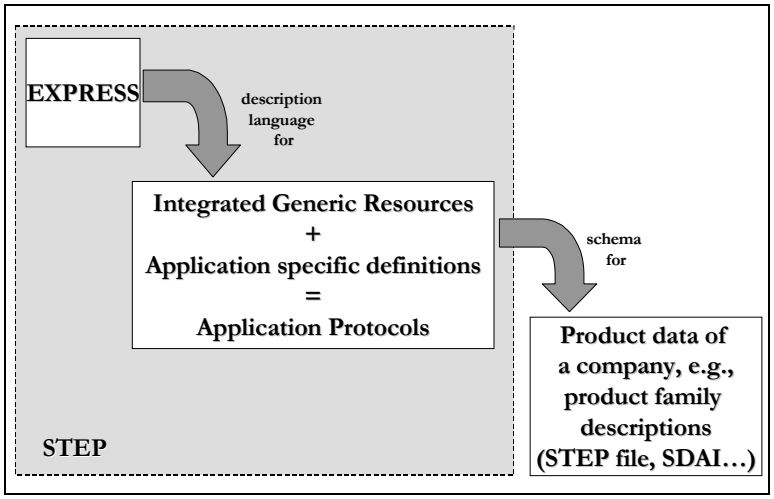

Figure 3. Relation of the main parts of STEP

Modelling of product families needs powerful concepts, such as the is-a relation with inheritance and conditions on the valid combinations of components. These were also introduced in AP214. The problem is that classes representing product families must be introduced as EXPRESS instances. Consequently, the inheritance between these classes is between instances. Moreover, these 'classes' and 'instances' are indifferent, that is, both the product family descriptions and descriptions of product individuals are similar EXPRESS instances.

In other words, product family descriptions are schema in their nature, as they need concepts such as the is-a relation with inheritance. In STEP, the schema becomes standardised. Product family descriptions, on the other hand, cannot be standardised since they are specific to each company and change frequently. This is how STEP is structured and thus AP214 needs to work around it by modelling classes as instances.

Therefore, it was concluded in Study I that without major refinement STEP does not suit well for product family modelling. Study I also proposed an alternative solution, which is only briefly addressed in the next section, where the focus is more on the fundamental problem, which in the following will be called *conceptual mismatch*.

5.1.2 Conceptual Mismatch

Study III argued that there is a *conceptual mismatch* between the traditional data modelling methods and the modelling of product families. The argument is based on the need for more conceptual levels in modelling product families than that provided by traditional data modelling methods. The modelling of product families and their individuals requires the following conceptual levels:

- *Basic concepts* are the modelling basis containing, for example, 'class', 'attribute', 'generalisation', 'inheritance', etc.
- *Product data modelling concepts*. On top of the basic concepts, one needs some concepts for modelling products. These are concepts not available as basic concepts, but are common to many product data modelling tasks. They include, for example, 'component', 'has-part relation', 'optional part', 'alternative parts', etc. In STEP, these are standardised in detail.
- *Product family descriptions* constitute the description of the product families of a company including class hierarchies of products and components. These are the descriptions of the product

families unique to a particular company, for example, different types of elevators and variation possibilities designed for them. In product configuration, a product family description is called a configuration model.

- *Product individual descriptions.* Finally, descriptions of product individuals are created according to product family descriptions. These are the descriptions of physical product individuals manufactured and delivered to the customers.

The term *traditional data modelling method* refers to methods based on the three conceptual levels of abstraction (Ullman 1988, Batini et al. 1992):

- *Concepts* (or *language*) contain the basic elements for constructing models about the *world* (or *universe of discourse*).
- *Schema* (or *entity types* or *classes*) is the model of the world constructed using the concepts.
- *Individuals* (or *instances, objects, data* or *entities*) are representations for entities that exist in the modelled world.

The conceptual mismatch thus refers to the fact that modelling of product families needs the four above mentioned conceptual levels and traditional data modelling approaches have three.

In product modelling, the problem of conceptual mismatch, however, becomes most relevant with large product variety, which is managed as product families. This statement is justified by the following comparison with mass products and project products.

For mass products, the product data modelling concepts can be introduced as a traditional schema. Product structures, i.e., BOMs, are then individuals of that schema. This leaves no level for the product individuals, which however, is not a problem since the descriptions of product individuals are typically not kept. If the records of individuals are needed, they can easily be obtained as copies of BOMs and then modified to reflect the 'as-manufactured' or other life-cycle status of the individual. In modelling for the design phase, the situation is similar, as there are no individuals yet.

Large project products, such as power plants and ocean cruisers are too complex to have a complete model, i.e., a product family description or equal, from which they could be instantiated.

In modelling of product families, the product family descriptions have the characteristics of the schema. Product individuals are instanti-

ated according to that schema and the descriptions of them need to be stored separately as each individual is typically different.

The conceptual mismatch explains difficulties traditional product data modelling approaches such as STEP have with product families (I). The problem is fundamental in nature and has various reflections on the modelling of product families and their evolution. It necessitates rethinking the role of the schema and consequently some elaboration on research results, for example, from schema evolution of databases.

One solution to the conceptual mismatch is to combine the product data modelling concepts with the basic concepts, and consequently, there would be only three levels. Many modelling efforts in product configuration actually work in this direction, as they search for the appropriate concepts for modelling product variety. This, however, easily leads to specific models for each application area or very large models in the lines of STEP. If the systems treat product family descriptions as schema, changes to the schema must be communicated between systems. Manufacturing, however, may not need full schema information, i.e., it can operate with concepts in which product structures simply consist of components. If the schema, however, needs to be transferred to after-sales, the class-information must either be embedded into the manufacturing instances or communicated pass the manufacturing.

Study I, proposed a solution in which the company specific schema was modelled as an extension to a standard schema. In practice, this requires, for example, that CAD systems need to be tailored to understand the schema of the company or they should have a general mechanism for reading in schema extensions. In both cases, it would be possible to work internally within the company using the specific schema. Transferring the product data, however, requires either tailoring the CAD system of the receiver if the systems cannot read schema extensions. Tailoring the CAD systems makes the propagation of changes to the schema tedious, as reprogramming may be needed each time the product changes. Therefore, it would be preferable to have systems capable of treating at least the schema extensions as data. Another possibility for transferring the data is to convert it to a STEP application protocol. If the company specific schema contains only specialisations of the application protocol concepts, the mapping is straightforward to implement—all individuals are just represented according the application protocol concepts. That is, all company specific definitions would be ignored and the conversion thus loses information. The EXPRESS-X mapping mechanisms can also be used

for implementing the conversion. It has the advantage that the data need not be imported to a CAD system for conversion, as conversion can be carried out with EXPRESS-X tools. The approach proposed in Study I thus sets requirements not only for STEP standardisation but also for the computer applications of the company, which need to be more flexible in adapting to new schemas.

Whichever the solution, the person making the decisions, should be well aware of the conceptual mismatch. Moreover, product family descriptions of a company would still be schema and the non-static role of the schema remains an open issue, as does the non-strict conformance of product individuals to the schema.

These observations need to be taken into account in definition of the concepts for modelling product families and their evolution. The approach taken in this thesis is to assume some very primitive basic modelling concepts, such as object and relation, and with them construct the modelling concepts for product families. However, since the focus is here on the aspects related to evolution, only the product family modelling concepts most relevant in capturing evolution will be defined.

As has been already argued, there should, in addition, be some support for semantics that reflects the evolution of products. This problem setting is explored and discussed in the following with the intention of finding some conceptual grounds for the needed support in advanced PDM systems.

5.2 Evolution Processes of Product Families

For the rest of this thesis, Figure 4 provides an overall picture of the evolution processes related to product families. The figure shows the schema and product individuals that are created according to the schema. The creation process is called instantiation, or also configuration process.

For the schema and individuals, the figure shows separate evolution processes. The evolution of the schema results from the changes made to the product. The evolution of a single product individual reflects the after-sales modifications made after the individual was manufactured and delivered to a customer (Männistö et al. 1996).

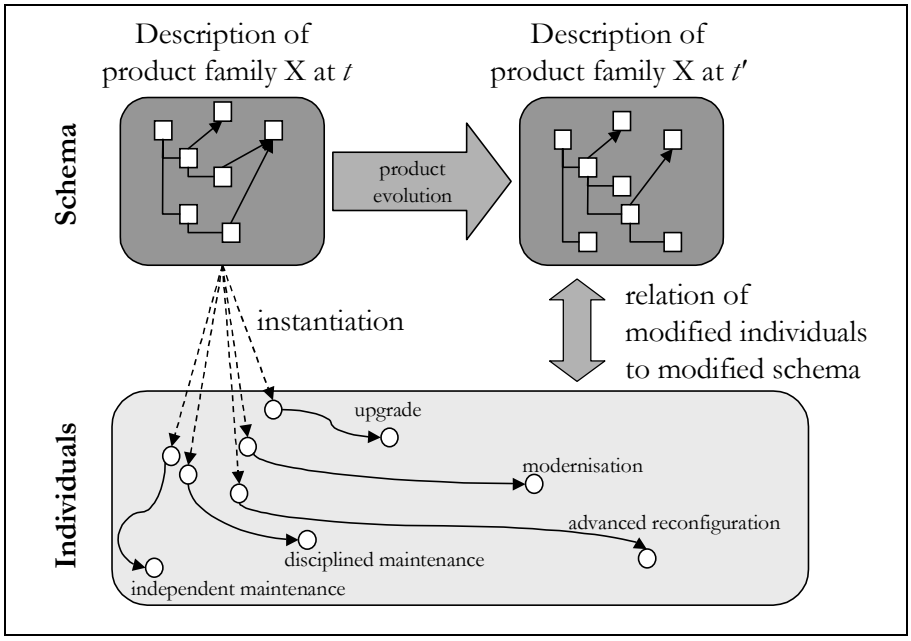

Figure 4. Relation of the schema and individuals [adapted from (Männistö et al. 1996)].

In addition to the evolution processes, interesting questions arise in the relation of product individuals to the schema when time passes. What can be done with old individuals when the schema has changed? How does a modified individual relate to the schema? How should individuals that are different and have no common schema be managed? These questions will be addressed in the rest of this thesis.

First, however, the instantiation process is investigated concentrating on the evolution of an individual within the process. The instantiation is not necessarily instantaneous, but can be a gradual refinement from many possibilities, i.e., a configuration model, to a description of a single variant.

Thereafter, the evolution of schema and individuals is addressed. The goal is to conceptually understand the whole in which the schema and individuals evolve rather independently, but still the relation between them is important. That is, the manufacturer does not forget the individuals as soon as they are delivered to customers.

5.3 Instantiation Process

Configuration, i.e., instantiation[2], is a process in which a description of a product individual fulfilling the requirements of the customer is created from a configuration model. A configuration process links the schema to product individual descriptions, i.e., the configuration model to configurations. The configuration process, however, is not necessarily instantaneous, as the customer order specification may gradually be refined towards the final configuration. The configuration process can be seen as a series of refinements removing all the variability expressed in the configuration model. The result of the refinements is an unambiguous description of a product individual to be manufactured. The other important aspects of configuration, such as backtracking and advanced search for the solution, are beyond the scope of this thesis (see, e.g. Peltonen et al. 1998, Soininen et al. 1998).

In a gradual configuration process, it is not clear where to draw the line between the schema and instances. For example, consider Figure 5. Without going into details, the basic notation in the figure is as follows. Boxes represent objects (there are no classes, only objects) and lines between them the specialisation relation (so that a lower object is a specialisation of the upper one). Each object has a name (on top of the box) and properties (below the name). Object properties include attribute declarations, e.g., "speed: float", attribute assignments, e.g., "speed ← 1.0" and constraints on attribute values, e.g., $\{0.6 \leq \text{speed} \leq 1.6\}$.

In the strict class-instance paradigm, the bottom-most entity, i.e., 'order 123/actual configuration', would be an instance and the others would be classes. This would be awkward, for example, with respect to the entity 'order 123/specification', which is not a part of the configuration model, i.e., it should be an instance. However, in an order specification it is, for example, possible to constrain the attribute values, not only to assign them. This would make the 'order 123/specification' more a class than an instance.

The point is that it is impossible to draw a clear line between classes and instances (or the schema and individuals according to the main terminology of this thesis) in a gradual configuration process. Therefore, in Study II an approach that abandons the clear distinction between

---

[2] In Study II, the term *instantiation* was used to mean the freezing of the result of a configuration process. Here *instantiation* means the configuration process of creating an individual from the schema, i.e., the whole process.

classes and instances was utilised. The study covers the evolution of a configuration inside the configuration process and gives directions for future work in modelling the evolution of schema and individuals with versioning of uniform objects, which was then elaborated in Study III.

The approach of Study II provides the required flexibility to class-instance paradigm for the configuration process. Abandoning the class-instance paradigm is justified for the time of the configuration process. However, the extension of the ideas to the evolution of schema and individuals is not trivial. It was already stated in Study II that although the model contains only objects, some objects are more class-like and some more instance-like. Consequently, it is meaningful to conceptually

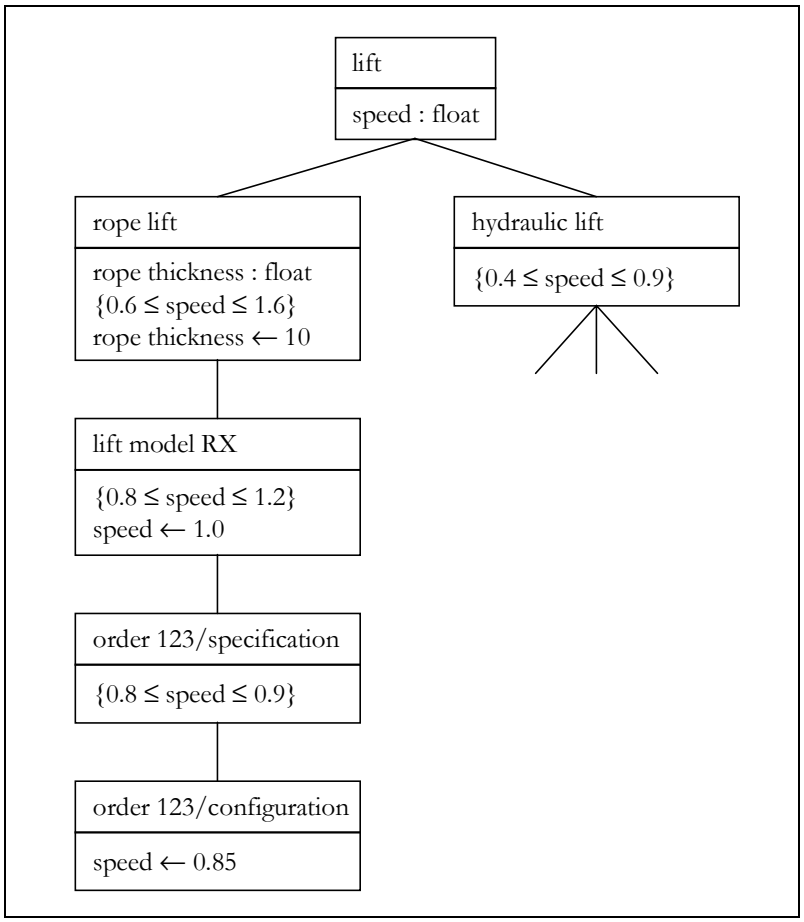

Figure 5. Example of a configuration process as refinement (II).

distinguish between the configuration model and the final representations for physical product individuals, i.e., the schema and individuals, respectively.

5.4 Evolution of Schema and Individuals

In Figure 4, there are the two evolution processes, namely the evolution of schema and the evolution of product individuals. With them as a problem setting, a question arises: "would the solution be a temporal database with the support for schema evolution?"

As was already discussed, there is an important difference between the role of the schema for product families and databases. The main distinction, as was stated in Study III, is the independence between the schema and individuals. Firstly, schema modifications are not propagated to product individuals. For example, when a new component (type) is taken in use in new products, the existing product individuals are normally not affected. Secondly, not all modifications to an individual are captured by the schema. For example, when a component is replaced in a product individual, the resulting individual contains components of old and new component types. Most probably, there has never been a single schema (version) with all component types of the modified product individual. Therefore, it would have been impossible to manufacture such an individual.

The independence thus means that: 1) individuals are not converted to reflect schema changes, 2) individuals of multiple schema versions may coexist and 3) an individual does not necessarily conform to any schema version. These points clearly contradict the data representation paradigm of databases, in which the schema strictly describes the individuals. Therefore, the evolution of product families needs something else that can directly be provided by a temporal database with schema evolution.

The solution proposed in Study III will be explained next. In addition, some aspects of the model will be elaborated further than what was possible within the limited length of the original paper. The solution has the goal of defining a conceptual platform for the evolution of the entities needed in modelling product families. The point is not to provide only the concepts, which have been investigated widely elsewhere, but also to incorporate the semantics of evolution so that the product data management policies of the company could be captured in advanced PDM systems.

5.4.1 Concepts

In this section, the concepts for the model in Study III are explained in a quite condensed form. The model is based on uniform *objects* and, in addition to objects, it includes *relations*. The relations are binary and are also called *references* from the first object to the second. To capture the evolution, the model uses *generic objects* similar to those discussed in Section 2.6; each generic object contains a non-empty set of *versions*. Each oval in Figure 6 represents a generic object and the small circles inside a generic object are its versions. Versions capture the evolution of a generic object and thus each version has *effectivity*. The *effectivity of a version* is an *interval*, shown next to the circle. For the clarity of illustration, the time axes of different generic objects are not aligned, but within a single generic object the time increases horizontally to the right. For simplicity, the versioning only captures the history of a generic object, therefore, no branching and no effectivity gaps are allowed. In other words, the effectivity intervals of consecutive versions of a single generic object are required to meet, i.e., there is no time instant between them and they do not overlap (Allen 1983). The *effectivity of a generic object* is the union of the effectivities of its versions.

The evolution of generic objects is modelled in a strict versioning sense. This means that all changes are implemented by creating new versions; versions themselves are not modified. That is, the *creation* of a generic object implies the creation of its first version, the *change* of a generic object leads to the creation of a new version and the *deletion* of a generic object means that the effectivity of its current version is ended. It is important to keep this usage of the terms in mind when reading this section.

Modelling all objects in products by generic objects provides a simple and powerful means for capturing the evolution of them as independent entities. That, however, is not enough since it is also very important to capture how the generic objects and their versions relate. Perhaps the most important relation in modelling products is the has-part relation, which is therefore included in the model. Another abstraction mechanism that is especially relevant in modelling product families is generalisation. It allows the modelling of abstract concepts, such as 'car' and 'motor', which are generalisations for the actual component types, such as 'sedan' and 'Model XY 2.0L'. Finally, the relation between the schema and individuals, i.e., the is-instance-of relation, differs from its counterpart in traditional data modelling, and

therefore, it will be represented explicitly. There are many other relations that could be included in the model, such as connections of components, physical location, compatibility, just to mention few. In order to keep the model relatively simple, the three first mentioned relations provide a small set of fundamentally important relations that was selected for further investigation. Inclusion of other relations is left as future work.

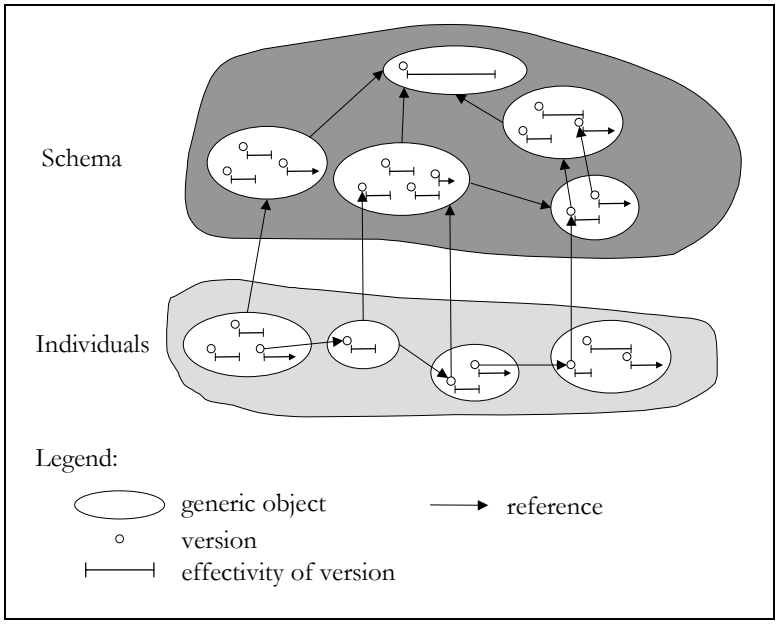

Figure 6. Overview of the modelling concepts (III).

The arrows in Figure 6 are examples of relations between various objects. The interest in the work was to capture the evolution of the three above mentioned relations. To sum up, relations are: the *is-a* relation with inheritance between types, *has-part* relation both between types and between individuals (but not between a type and an individual) and *is-instance-of* relation between individuals and types. Of interest is also *conformance*, that is, whether an individual is a valid representative of its type.

Objects in the schema are called *types* and individual objects are simply called *individuals*. Each type and individual is either a generic object or a version. Consequently, the lowest level objects in the taxonomy of

concepts shown in Figure 7 are *generic type, type version, generic individual* and *individual version.*

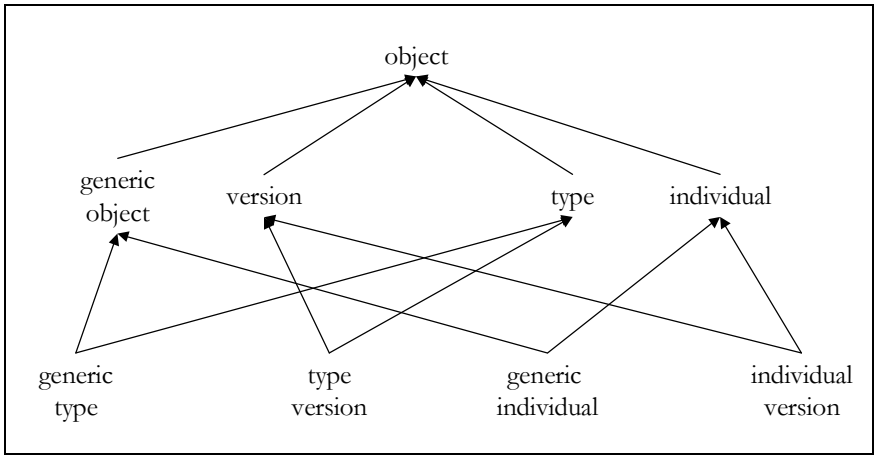

Figure 7. Taxonomy of concepts.

Relations (i.e. references) are allowed between all kinds of objects, e.g., from individual version to generic type or from type version to type version. There can be *generic references* and *fixed references*. A generic reference is a reference to a generic object and bears the intention to be bound to one of its versions. A fixed reference, on the other hand, specifies the version, and thus, no resolution is needed. A reference from a generic object concerns all its versions, i.e., they all have the reference, whereas a reference from a specific version concerns that version only. References, once set cannot be modified—a reference from a version can be effectively changed by creating a new version with a new reference. The semantics of these will be discussed shortly, but before that the invariants for capturing the evolution of different objects with respect to the above-mentioned relations are presented.

5.4.2 Invariants

The concepts presented thus far enable capturing the evolution of objects that represent product families and product individuals. There is yet nothing for the semantics needed in product family modelling. The concepts can be used in various ways, and therefore, in Study III, invariants were defined to constrain the usage of the concepts. The invariants were defined separately for each relation, and the goal was to

find a combination of invariants that meaningfully captures the evolution of product families and their individuals. That is, by fixing the concepts and considering different ways of constraining their usage, the intended semantics can be approximated.

The invariants defined in Study III are repeated here with brief explanations of their intended semantics. Details concerning generic references will be discussed thereafter.

Is-instance-of relation

Is-instance-of is a relation between individuals and types. For an individual, it denotes the type of the individual (if any). The validity of an individual with respect to its type is controlled by means of conformance. Two invariants were defined, a strong and a weak.

Strong conformance invariant: *An individual is constantly kept in conformance to the type it is-instance-of.* (**1**)

Invariant 1 requires that at any given time *t* the effective individual version relates by is-instance-of and conforms to exactly one effective type version. For modelling product families, however, this is too strict. Therefore, a weak conformance invariant was also defined.

Weak conformance invariant: *The first version of a generic individual conforms to the type version it is-instance-of at the time of its creation.* (**1'**)

When a generic individual, i.e., its first version, is created, it must conform to its type, but may thereafter evolve independently. This reflects the situation with product individuals that are manufactured and then delivered to the customers, where they may evolve rather independently of the manufacturer.

Is-a relation

Is-a is a relation in the schema that organises the types into taxonomy, and provides a mechanism for sharing common properties by means of inheritance. Two invariants were defined for controlling the effects that changes have via inheritance. Nothing in the model actually prohibits multiple inheritance, but for simplicity, only single inheritance is considered here.

Strong effectivity invariant for is-a: *Effectivity of each (sub)type version must be contained in the effectivity of the single version of its supertype.* (**2**)

Consequently, a new version may be created for a generic type without a need to update its supertypes, but for its subtypes, new versions need to be created. Invariant 2 guarantees that the inherited properties of a type version remain unchanged. It provides a strict modelling basis in which the properties of a type version never change during its effectivity.

Existence of supertype invariant: *Effectivity of type must be contained in the effectivity of its (super)type.* (**2'**)

Invariant 2' is a weaker form of Invariant 2. The main difference between them is that Invariant 2' only requires that an effective version for the supertype exists, not that the changes in it are propagated downwards. Consequently, the inherited properties of a type version may change during its effectivity.

Has-part relation

Invariants defined for the has-part relation are similar to those of the is-a relation.

Strong effectivity invariant for has-part: *Effectivity of each version must be contained in the effectivity of single version it has as part.* (**3**)

Existence of part invariant: *Effectivity of an object must be contained in the effectivity of an object it has as part.* (**3'**)

The invariants are applicable to has-part between types as well as between individuals. For has-part between types, Invariant 3 corresponds to the versioning semantics in which a modification to a component is propagated to all composites using it as a part. Invariant 3' corresponds to the semantics in which components may evolve independently, typically as long as changes are internal to the component.

5.4.3 Generic and Fixed References

The invariants were given above without discussing generic and fixed references. Although, resolving generic references has been widely discussed in the literature, here the references are in a new context in which it is not immediately clear what they mean. The role of generic references in capturing the semantics of evolution is central, and therefore, their relation to invariants will be discussed in detail in this section.

A reference from a version or to a version is clear in the sense that it is fixed to the version. The references from and to generic objects deserve a more detailed discussion.

A reference from a generic object concerns all versions of the generic object. That is, each version has the reference, including also the future versions of the generic object; the reference has been predetermined for them.

A reference to a generic object is a reference to the version set with the intention that the reference will be bound to one of the versions when needed. How and when this resolution is made is the main semantic consideration of these references. One resolution mechanism with effectivity intervals is global time. That is, given a time *t*, all references to generic objects are bound to the versions effective at *t*. This semantics was also used in Study III, with the exception of the weak conformance invariant (i.e., Invariant 1'), which speaks of the creation time of the generic individual. Therefore, an is-instance-of reference to a generic type becomes bound to the type version effective at the creation time of the generic individual when conformance is inspected.

The cases for different kinds of references with the invariants are illustrated in Figure 8 and commented on in Tables 1–3. In the figure, each case is represented by a pair of generic objects. In each pair, effectivities of versions are aligned vertically. Labels identify the cases so that S stands for *strong* (i.e., Invariants 1, 2 and 3), W for *weak* (Invariant 1') and E for *existence* (Invariants 2' and 3'). The pair of letters right of the colon denotes the reference kind, e.g., "g-v" means a reference from a generic object to a version. For example, in the S:g-g case of is-a, the upper generic object is a superclass of the generic object below, the solid arrow represents the is-a relation between the generic objects and the dashed arrow the propagation of the change when a new version is created for the superclass.

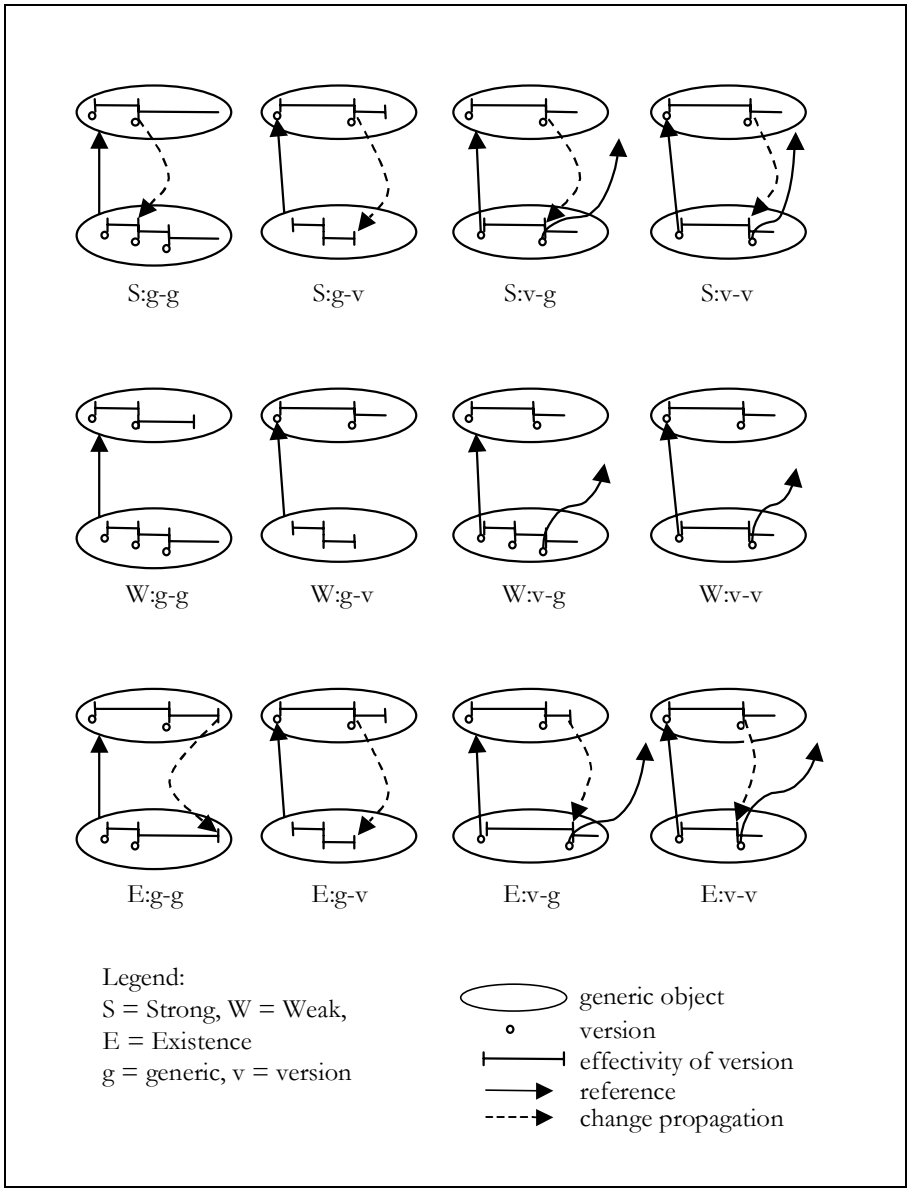

Figure 8. Cases of different kinds of references and invariants.

Table 1. Reference cases for the is-instance-of relation with conformance invariants. (In Figure 8, the lower object is the individual and the upper one the type.)

Case	**Comments**
S:g-g	At any time, the effective individual version conforms to the effective type version. If a new type version is created, a new individual version needs to be created to retain the conformance. That is, a schema change is propagated to individuals. (Actually, the invariant would not necessitate the propagation if the change implied conformance. Such changes are ignored for simplicity.)
S:g-v	Each version of the individual must conform to the referred type version. New versions can be created for an individual as long as the conformance is maintained. A change to the type necessitates deletion (logically, not physically from the database) of the individual, i.e., the effectivity of its current version is ended.
S:v-g	As the reference is from an individual version, a new individual version must specify its type, but this can be done independently of the previous versions. This case allows the conversion of an individual into a different type by creating a new version for the individual.
S:v-v	Same as S:v-g.
W:g-g	The first version of an individual must conform to its type (i.e., the type version effective at the creation time of the individual). Thus, an individual is created according to its type, but thereafter no conformance is required. Both the type and the individual can evolve freely. There must be an effective type version at creation time of the individual.
W:g-v	Same as W:g-g, except that the conformance must be to the referred type version, which need not be effective at creation.
W:v-g	For the creation of the individual as W:g-g. However, a conversion of an individual can be recorded as a reference from the converted version to the new type (version). Conformance, however, is only required for the first version of the generic individual.
W:v-v	Same as W:v-g, except that conformance must be to the referred type version.

Table 2. Reference cases for is-a invariants. (In Figure 8, the lower object is the subclass and the upper one the superclass.)

Case	Comments
All strong	Each case differs from the respective one in Table 1 only by the detail that the definition is based on effectivity containment instead of conformance. The difference is that effectivity containment requires propagation, that is, whenever a generic object is changed or deleted, that change must be propagated. (See also the comment on S:g-g in Table 1.)
E:g-g	It is only required that the union of effectivities of a type contain the union of effectivities of its subtype. Thus, a deletion of a type must be propagated to its subtypes by deleting them.
E:g-v	Same as S:g-v.
E:v-g	The deletion of the type requires creation of a new subtype version, which may become a subtype of another type. The is-a relation from a (sub)type can be changed any time by creating a new type version.
E:v-v	Same as S:v-v.

Each case for has-part is similar to the respective is-a case. Therefore, only the effect is summarised in the following table with terms of part relations. The figure, however, should be read so that the whole is (awkwardly) the lower object and its part the upper object.

Table 3. Reference cases for has-part invariants.

Case	Comments
S:g-g	A change in a component is propagated to all objects that have it as part, which then become changed.
S:g-v	A change in a component implies the deletion of the whole!
S:v-g	The change of a component requires the change of all wholes using it as part. The reference is from a version and thus a new version of the whole must specify its part, although independently of previous versions.
S:v-v	Same as S:v-g.
E:g-g	The deletion of a component must be propagated to the wholes by deleting them.
E:g-v	Same as S:g-v.
E:v-g	The deletion of a component requires a new version for the whole, which then re-defines its parts. Parts of the whole can be changed any time by creating a new version for it.
E:v-v	Same as S:v-v.

5.4.4 Conceptual Platform for Evolution of Product Families

In Study III, three kinds of invariants were defined: strong, existence and weak. For a traditional database, the strong invariant would be appropriate for all cases. For capturing the evolution of product families and the individuals that is too strict. For them, Study III thus proposed the following selection of invariants as a "normal" selection:

- weak conformance invariant for is-instance-of (i.e., Invariant 1')
- strong effectivity invariant for is-a (i.e., Invariant 2)
- existence of part invariant for has-part (i.e., Invariant 3')

This selection of the conformance invariant allows rather independent evolution of the schema and individuals only requiring that individuals are created according to the schema. Invariant 1' also allows the recording of individual conversion (when reference cases W:v-g and W:v-v of Table 1 are used). The invariant only requires the conformance for the first version of an individual; thus, recording the type for other individual versions bears no semantics, but can act as an annotation. In reality, however, there hardly exists a schema version to which the individual could be converted. The relation of modified individuals and the schema was addressed in Studies IV and V, which will be discussed later.

For conformance with a weak invariant, the most appropriate kinds of references would be W:g-g and W:v-g. The other two differ from these only slightly and the references to generic type are preferred here as they leave the resolution for other possible aspects open, such as multiple versioning sequences or branches within the versions of a generic object—these, however, are beyond the scope of this thesis.

For is-a, strict change control was required. This is to prohibit changes in the inherited properties of a type version during its effectivity. That is, if any of the supertypes of a type is changed, the change must be propagated to the type (and its subtypes), which means that new type versions with new effectivity intervals needs to be created.

Of the kinds of references, S:g-g is probably the most useful one for a generalisation hierarchy. It ties all versions of a type to the same supertype. If that is not wanted, i.e., versions of a generic type may have different supertypes, S:v-g could be used. Case S:g-v seems odd since it only allows the subtype to live for the effectivity of a single supertype

version. Case S:v-v is same as S:v-g and the latter is preferred for the same reasons as was discussed with conformance.

Has-part is an important relation in products. It has rich semantics in companies and there are change processes and policies for controlling evolution in product structures. One particular issue is the substitutability of component versions, which typically means that a new component version can be used in place of older versions of the component. If substitutability cannot be maintained, a new component must be created instead of a new version. From this principle it follows that a change in a component need not be propagated to the wholes using it as part. The invariants for has-part were defined and selected to capture this particular semantics of evolution.

With the reference kind E:v-g it is possible to version the component without change propagation, but the deletion of the component must be propagated. The case also allows the whole to change its parts. The reference kind E:g-g is stricter because all versions of the whole have the same generic object as part. E:g-v is again somewhat odd since once the component is changed the whole must be deleted. E:v-v requires propagation of all component changes.

This framework for supporting the evolution of schema and individuals contains a selection of invariants and guidelines for the usage of different kinds of references. Simple semantics are already included, but in an advanced PDM system, there should be a mechanism for selecting various constructs for the needed semantics, which could also vary between product lines. The framework provides a platform that needs to be refined by the systems. However, as it stands, it is already a small breakthrough since it shows how the semantics of evolution of product families and their individuals could be captured in product data management systems.

The results presented thus far are on a rather high level of abstraction. In particular, the evolution of individuals of product families is far more complicated than what is accurately captured by the given form of weak conformance. In the following, solutions for capturing the evolution of individuals were searched from the after-sales viewpoint.

5.5 Relation of Evolving Schema and Individuals

As has already been pointed out, the schema and individuals of product families differ from their counterparts in databases. The main distinction is the independence of an individual of the schema after the creation of

the individual. The individual, however, is not entirely independent of the schema. After-sales modifications of a product individual typically utilise component types introduced in newer schemas. Next, a closer look is taken at the world of product individuals.

The world of product individuals is illustrated in Figure 9. The whole picture represents all imaginable product structures, i.e., configurations. Areas within the figure represent various sets of (potential) individuals from the modelled world. The sizes of areas are not proportional to the number of individuals contained.

In Figure 9, the largest area drawn with a thick solid line represents all meaningful products. That is, configurations of components that are meaningful in the sense that they form a functional product that has specific usage. Inside the meaningful configurations are areas drawn with a thin solid line. These areas represent configurations that are valid according to a schema, i.e., a configuration model, at a particular time.

These areas are within meaningful configurations, since a configuration model is required to represent only functional products. It is, however, not required to represent all meaningful configurations achievable from the component types of the company. That is, the approximation made by a configuration model is a subset of all meaningful configurations that can be made of the component types of the company.

The point in Figure 9 is to illustrate how product individuals are represented in the different modelling approaches addressed in the studies of this thesis. The figure thus combines into one picture the results of the Studies II, III, IV and V.

Study III focused on the modelling of evolution with an emphasis on the independence of the evolution processes of the schema and individuals. The weak conformance only requires that individuals are created according to the schema. In the figure, this means that the dot representing an individual initially appears within the area representing the schema of the creation time of the individual. The configuration process creating product individuals was addressed in Study II.

The evolution of an individual, which was represented by versions of a generic individual in Study III, is represented in the figure by arrows. After a change, an individual may remain within the initial schema, may be represented by another schema version or the modified individual may be no longer represented by any schema version. It is also possible that an individual ceases to be a meaningful configuration if, for example, an essential component is removed from it.

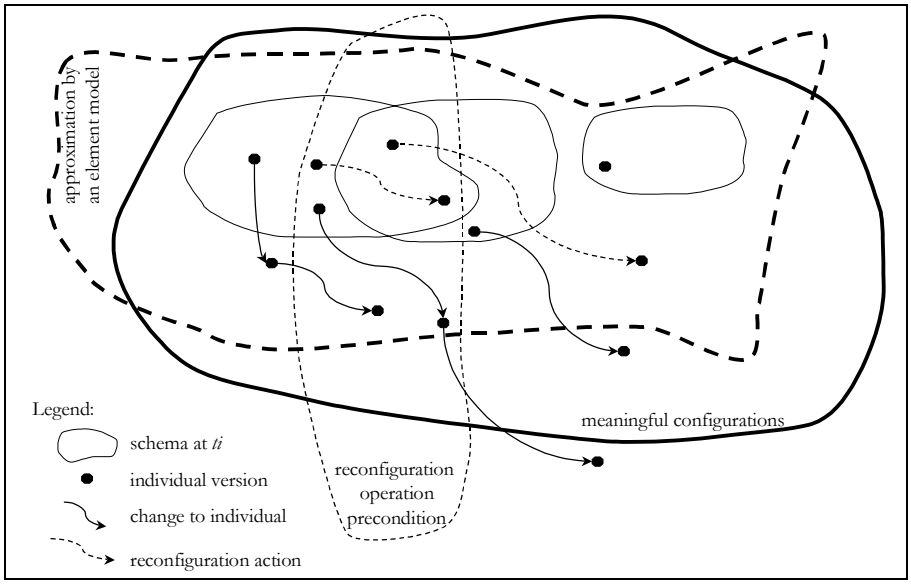

Figure 9. Approximations of the population of product individuals.

In Study III, the versioning of objects in the schema induce schemas of different times, i.e., areas 'schema at *ti*' shown in Figure 9. The chains of arrows represent the evolution of individuals, which in Study III was represented by individual versions independently of the schema. Studies IV and V were carried out for finding means for capturing the evolution of the individuals more appropriately.

Study IV modelled individuals to which a reconfiguration operation can be applied by a reconfiguration precondition. A precondition is exemplified in Figure 9 by an area drawn with a thin dashed line. In principle, the source individual to a reconfiguration operation may be non-functional, e.g., broken. In such case, however, reconfiguration operations are not required to fix a broken individual since in Study IV the aim was to provide means for modelling reconfiguration operations that keep originally valid individuals within the boundaries of meaningful configurations.

The purpose of Study V was to provide means for uniformly modelling all product individuals that are serviced by the company. The population represented by the model must thus include all product individuals configured by the company and some others. The purpose of the model, unlike a configuration model, is not to provide exact manufacturing instructions for a new product individual. Its purpose is

to identify some aspects from existing (working) product individuals. Therefore, the approximation of this model is by a superset, as it is more important to represent all existing individuals than not to represent any non-functional ones. Studies IV and V will be dealt with in turn below.

5.5.1 Reconfiguration

The instantiation process creates a description of a product individual from a product family description. Reconfiguration, on the other hand, refers to the process in which an existing product individual is modified to meet new requirements of the customer. Reconfiguration is an important business for companies that offer customer-specific solutions. Moreover, it is importance is increasing with the general trend towards better customer satisfaction.

Reconfiguration is in many respects a complicated business. However, some trends make it more desirable than before. For example, extending the lifetime of product individuals can be motivated by environmental arguments and extensible products are attractive when investment budgets are put under tight scrutiny. Reconfiguration is problematic because it cannibalises the markets from new products. Maintaining a high degree of reconfigurability may also hinder product development by making it harder to introduce new technologies into products.

Reconfiguration is not equally feasible for all kinds of products. For very complex products, such as power plants, it may be difficult to define systematic reconfiguration knowledge, and therefore, reconfiguration must be carried out as projects. Mass-products, on the other hand, may be too cheap, so that it is not economically feasible to reconfigure them—as it is cheaper for a customer to buy a new product. Therefore, from different kinds of products, the best candidates for reconfiguration are configurable products.

Another factor that affects the feasibility of reconfiguration is the length of the lifetime of product individuals. Product individuals that have a long lifetime and high cost are typically investments. It is very likely that such products are modernised at some point in their lives. In addition, although reconfiguration would principally be feasible for some products, the high rate of technological change may make it unfeasible. For example, personal computers become quickly outdated and thus modernisation would require changing almost every component, which is more expensive than buying a new computer.

Reconfiguration typically happens some time after the product was originally configured, when both the schema and the product individual have probably changed. Therefore, reconfiguration is an issue of the relation between modified schema and modified individuals of Figure 4. The old individual as input makes reconfiguration more difficult than configuration. For example, something that does not affect the actual configuration process may be a critical input to reconfiguration, such as the support structures in the installation environment.

There are several ways how companies have handled the reconfiguration (IV). One is to initiate a project that investigates the old individual and designs and implements the modification. This is an expensive, but in many cases the only, possibility. Some companies have tried to find less costly solutions for reconfiguration in cases that are close to each other and rather frequent. One such solution is to define reconfiguration packages, which are pre-designed to be reusable. Another, more advanced, approach is to systematically design the product to be reconfigurable so that a significant portion of new features can also be offered to old product individuals. This approach lays a burden on the product development in the sense that it is more difficult to introduce new technology to the product. Nevertheless, the approach also ties customers more tightly to the company and gives customers the confidence that with small gradual investments they can keep their products functionally modern.

In Study IV, a conceptual model was defined to represent reconfiguration independently of configuration models. The model consists of *reconfiguration operations* and *reconfiguration invariants*. A reconfiguration operation is of form <*pre-condition, action*>. The idea is that a reconfiguration operation can be applied to a product individual if the pre-condition evaluates to true. The effect of applying a reconfiguration operation is described by the action, which tells how to change the individual. Reconfiguration invariants are conditions; it is required that the resulting product individual must fulfil the reconfiguration invariants. The model is based on the approach used by one industrial company for providing systematic reconfigurability for product individuals they have manufactured.

Figure 10 shows a practical example of reconfiguration operations. The schema changes according to product evolution, which is illustrated at the top of the figure. With a modified schema, it is then possible to consider how new components in the schema relate to old ones. Typically, some substitution rules are given. In the figure, for example,

the component types (motors) 97998 and 97999 are at some point replaced by a new one, i.e., 98070. In addition, the figure shows an upgrade operation, which allows certain 8 to 10 tons cranes to be upgraded to a 12 tons crane when preconditions are met. The approach thus allows capturing reconfiguration operations even when there are only few of them and extends to more systematic support of reconfiguration, possibly with a reconfiguration system for automatically validating or finding sequences of reconfiguration operations.

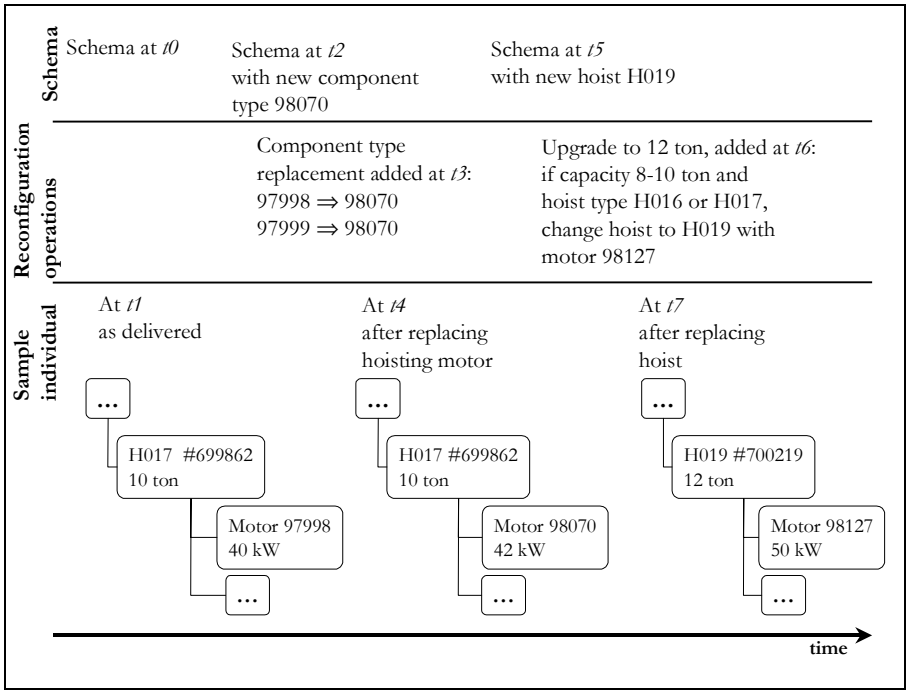

Figure 10. Examples of reconfiguration operations.

In addition, some notion of optimality is needed as otherwise a reconfiguration task could be solved by changing all the components, i.e., configuring a new product individual for the customer (which, however, may be most optimal in some cases). The solution with the cheapest components is also not necessarily optimal, although it might be a good candidate.

A configuration model describes only valid combinations of components—it must not allow any invalid configuration. For reconfiguration modelling, the approach taken in Study IV is weaker as it assumes that the starting point for reconfiguration is a valid product individual.

The same applies to reconfiguration invariants; they are set to guard against combinations of reconfiguration operations that may lead to invalid individuals. If the starting point is an invalid individual, the reconfiguration operations and invariants are not needed to detect or correct that.

In Figure 9, the area drawn with a thin dashed line illustrates individuals for which the precondition is true. The area also includes some non-functional configurations. Evaluating the reconfiguration action for a non-functional individual would most probably lead to another non-functional individual. Therefore, it was required that the starting point for a reconfiguration operation must be a valid individual.

In Study IV, it was argued that capturing the reconfiguration within the configuration model is unfeasible, the main argument being that the configuration knowledge was in all investigated companies very much surface knowledge. Consequently, incompatibilities between the components of different schema versions would not be automatically detected, because the reasons for such incompatibilities are easily abstracted away from the configuration models. Constructing a configuration model with deeper knowledge increases costs and, although it would reduce the problem, the model would nevertheless be an abstraction. The proposed model provides another approach for reconfiguration modelling. The model is derived from the experiences with industrial companies and it incorporates the methods used by the companies that supported reconfiguration.

Reconfiguration is flavoured by characteristics of after-sales. This is also reflected in the optimality criteria of reconfiguration. A company may want to reduce the number of components in service or to use the existing storage of old components. Such considerations may make a selection of more expensive components optimal. Sometimes it may also be wise to change more components than is actually necessary, for example, some cheap components that have a tendency of wearing out. Therefore, modelling reconfiguration separately from configuration models is also backed-up by the fact that configuration and reconfiguration are separate business processes, which leads to different requirements on the systems supporting them. The separation of reconfiguration and configuration tasks is also proposed by Stumptner and Wotawa (1998).

5.5.2 Modelling Product Family Individuals for After-Sales

Product individuals of a product family are rather different from each other because they are individually adapted to the requirements of each customer. This can be partly solved by keeping records of each product individual, e.g., 'as-manufactured', 'as-installed' or 'as-maintained'. With such information, details of each individual are known and the individual can be serviced and maintained accordingly. If a company wants to sell customers "uptime" instead of service hours after something breaks down, the company needs to known accurately the wear-out characteristics of products it maintains and have consistent records of all performed service operations. The heterogeneity of the individual population, however, still makes it difficult to acquire information about different component types used in similar functions or to distinguish between components of the same type used in different functions. For example, to estimate the wearing out of brakes of end-carriages or to find out the number of times a certain motor type is used as a hoisting motor in cranes.

Representation of the individual population of a product family is a particular problem in modelling the relation between the schema and individuals of product families that was presented in Figure 4. Not only are the individuals different because of the initial adaptation to the customer requirements but also because the individuals are created at different times. That is, the individuals may be created according to different schema versions. Furthermore, the evolution of the individuals is not captured by the evolution of the schema—changes made to the product families do not describe after-sales operations for product individuals. Nevertheless, for after-sales a schema that would describe, even approximately, the heterogeneous population of individuals would be beneficial.

Study V developed a mechanism for defining such schemas with *element models*. An element model captures the essential elements of each individual in a standardised manner. Figure 11 shows a largely imaginary example in which the BOM structure of an individual is associated with an *element structure*. The idea is that an element structure contains general elements such as 'end-carriage' or 'hoisting motor'. To each element in an element structure, a component or components from the BOM are associated; these associations are indicated in Figure 11 by dashed lines.

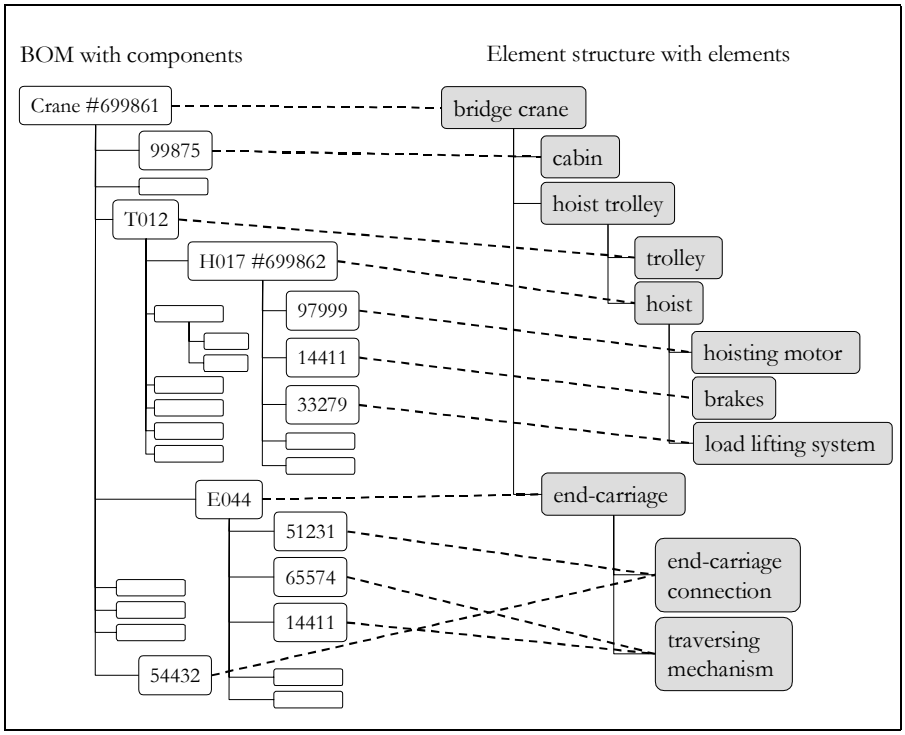

Figure 11. Example of a BOM associated with an element structure.

Because of the generality of elements, most BOMs could be described by a rather small set of elements. With the association to the elements in an element structure, it becomes easier to identify from a set of BOMs motors that have been used as hoisting motors. The element structure also serves as a point of reference, independent of physical components, for service operations. For example, it allows recording the change history for brakes of a hoisting motor. These records are retained even if the whole hoist is changed (as it is assumed that the element structure does not change during the life of an individual, at least not in a manner that would destroy history).

In order to be useful for analysing the population of product individuals, the element structures must be constructed in a standard manner. The number of elements should be small, the element structures for different product families should be as similar as possible and the association with the BOMs should be complete enough. The

standardisation of element structures was achieved in the study by element models, which define the allowed element structures.

Figure 12 (partially) shows examples of product individuals that have been manufactured at different times. Although the BOMs of the individuals use different types of components, their element structures are similar. This similarity is guaranteed by an element model. The element model has specified the element 'cabin' as optional, and therefore, not all element structures need to have a cabin.

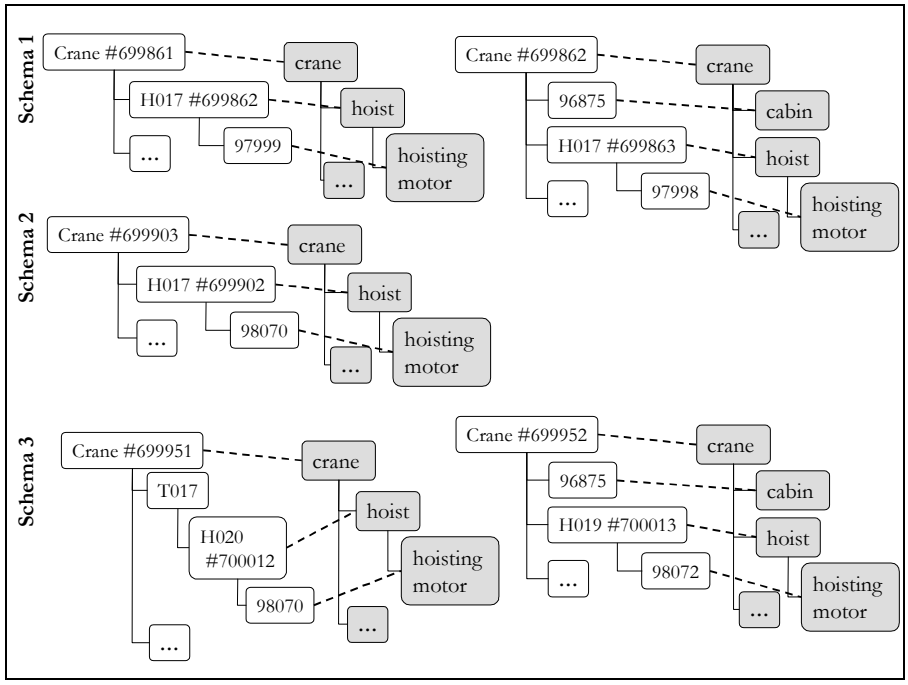

Figure 12. Sample product individuals with associated element structures.

The population of product individuals, however, is more complicated than described so far. Product individuals may have very long lifetimes, tens of years or so, and consequently, there is very little recorded information about the oldest ones. Moreover, it is typical that a company services product individuals manufactured by others, in which case, the information available may be incomplete or hard to find. Even in these cases, some information can be acquired on the product individual; its main parts, for example, can be identified. Nevertheless, the element structures for these individuals must be more general than

for the product individuals currently manufactured by the company itself.

For supporting the representation of product individuals at different levels of abstraction, Study V provided a mechanism for organising the element models in a specialisation hierarchy. Specialisation between models means that a model S is a specialisation of model M, if S represents a subset of the individuals represented by M. For example, if the most general crane model requires that each crane has exactly one hoist, each specialisation of it must have an element hoist (or its specialisation). Thus, it is possible to identify the components of the hoist in any crane individual.

The above definition of specialisation does not directly tell how to make a specialisation of an element model and it may be difficult to see whether that is the case by looking at element models. Therefore, a set of specialisation operations was defined for creating specialisations. The operations were also proved sound.

The element models proposed in this thesis originate from a co-operation with an industrial company and the problems they had in managing the population of product individuals in after-sales. The idea of the approach is to cover all the product individuals that are serviced by the company. These include old individuals manufactured by the company and individuals manufactured by others but serviced by the company. It is impossible to model such sets of individuals as strictly as new potential individuals are represented by a configuration model. The approximation of individuals by an element model is illustrated in Figure 9 by a thick dashed line. The idea is thus to create product individuals with a configuration model, let the individuals evolve independently and then utilise element models to represent them systematically for the purposes of after-sales. The schema of element models is thus independent of the configuration models according to which the product individuals are originally created. The company tested the approach by defining some three hundred element classes and during the research project implemented a prototype tool for modelling their products with the element classes.

6 Discussion

In this section, the thesis is compared with related work and product data management in industrial companies. The discussion begins with a general view to the fundamental issues and problems of conducting research in the area of industrial products and organisations.

Research on Industrial Products—General Remarks

Design and modelling of product structures and product families belongs to *the sciences of the artificial,* according to the term coined by Simon (1996). In contrast to natural sciences, the sciences of the artificial deal with artefacts constructed by humans. The world investigated is not stable as in addition to being constructed it is continuously modified by humans. This certainly applies to product families. The product families we are modelling today did not exist a century ago and most probably the product families of the next decade will be quite different from the current ones. Gravitation, on the other hand, does not change, although our understanding of it may.

Moreover, the product families of different companies need not have much in common. Nevertheless, there are some principles that can be applied to a wide range of different kinds of products, but, the more detailed the concepts, the fewer products there are to which they nicely fit. This leads to a dilemma as the most general concepts are very abstract and can be (mis)used in various ways so that their semantics become blurred from the product data modelling viewpoint.

In defining suitable concepts and semantics for product families, one is also faced with another problem. Namely, should the model reflect the reality or should one reach for better and cleaner concepts and suggest changing the reality to reflect them. Typically, the best solution seems to be somewhere in between. That is, the concepts should be derived from the real world but they need not exactly reflect the real world if it is too messy, which typically is the case.

Many of the difficulties in companies are independent of information systems, or more precisely, they cannot be solved by introducing an information system. An implemented system affects the company as it forces the operation within the concepts, semantics and functions supported by the system. If the choice of concepts is correct enough, an

information system may consequently help in improving the data management processes of a company.

The results of this thesis emphasise the conceptual aspects of product family modelling. The research has been conducted in a close contact with industrial companies. Study II stemmed from co-operation with a large elevator manufacturer, Study IV was based on the results from over two dozen Finnish companies manufacturing configurable products and Study V on co-operation with a crane manufacturer. Study I investigated the largest product data modelling standard. Study III then built a framework from the experiences of the other studies. Overall, the idea was to construct a holistic view to product families of industrial companies.

The purpose of this summary part is to provide a unified view on the separate original studies behind this thesis. The original studies were rather independent studies made at different times, in distinct research projects and with various industrial companies. Consequently, they do not form a natural whole with a consistent terminology. The terminology, for example, reflects the evolution of ideas during the research as well as the viewpoint taken in conducting a particular study and in reporting its results. Nevertheless, the original studies address the very same phenomena and thus exhibit notable commonality, which was then made explicit in this summary part of the thesis.

When the results of this thesis are evaluated for practical usage, for example, when considering their implementation in the product management processes of a company or in a PDM system, the relation of the studies to each other must be understood. That is to say, the studies should not be blindly combined; instead, a consistent whole should be looked for in the manner of this summary.

The implementation of the results of this thesis into a PDM system would require the system to be built relatively cleanly on an object-oriented basis and to provide an object-oriented model for modelling products. This, however, does not necessitate the underlying database being object-oriented. The implementation of the effectivity and generic version referencing mechanisms that are behind the proposed approach necessitates, in practice, representation of generic object separately of its versions. However, it may be that the basic concepts for modelling product families, e.g., generalisation, optional and alternative parts and conditions for modelling valid component combinations, set tougher implementation requirements for a PDM system than addition of the support for evolution in the proposed form.

One major new requirement for an existing PDM system is the keeping of records of product individuals. In fact, it can be argued how tightly the after-sales operations and basic product data management should be integrated. It may not be necessary to include them within a single system. Although if implemented in separate systems, the systems would nevertheless share a great deal of information, e.g., component types, and therefore, they need to be rather tightly integrated anyway.

The support for multiple abstraction levels in modelling the individuals is based on similar modelling concepts to the modelling of product families. The main distinction is in the approximation; i.e., only the validity of element structures is checked: The system does not need to generate individuals. For implementation, perhaps most challenging would be the support for specialisation between models. The proposed approach, however, provides operations for creating specialisations, which means there is no need to investigate whether a model is a specialisation of another model.

This is a general view of implementing the proposed concepts in a PDM system. Instead of addressing detailed implementation concerns, this thesis concentrated on forming the conceptual basis and raising the awareness of the phenomena related to the evolution of product families.

Conceptual Mismatch

The problem of conceptual mismatch (although not using this term) has also been discussed, in addition to this work, by Erens, McKay and Bloor (Erens et al. 1994, McKay et al. 1996). In their earlier paper (Erens et al. 1994), they raise the awareness with respect to abstraction levels in the modelling of product families. In their later paper (McKay et al. 1996), they illustrate their idea with an intermediate conceptual layer (called Layer 2), which resides between an EXPRESS definition of product data modelling concepts (Layer 1) and EXPRESS instances (Layer 3). Layer 2 contains the product family description, which can be regarded as an instance of Layer 1. Similarly, the individual description at Layer 3 is an instance of the product family description at Layer 2. As a result, Layer 2 has a data model view and instance view within it. The exact relation between these views, however, is somewhat vague. In addition, the use of is-a relation for modelling product families is not discussed. The model, nevertheless, is an important contribution towards solving the conceptual mismatch problem and the authors conclude with the wish that such

requirements would be catered for in engineering databases of the future (McKay et al. 1996).

Semantics and Conformance

With global operations, thousands of employees and large amounts of information, the management of product data necessitates some formal procedures. Such control is a delicate matter, as routine processes should be strictly controlled by systems and innovative processes need more freedom. There are tales that CAD systems destroyed old release and approval procedures in some companies as the manager's approval signature was already scanned in the drawing templates. PDM systems remedied this situation by taking the control of the approvals.

Most PDM systems support versions but are typically less explicit in capturing the evolution semantics of controlled objects. Product change processes are typically managed in the form of work-flows. In an engineering change, a set of modified entities can be grouped under the change order and so handled as one set. The propagation of a change in product structures is typically controlled by some means, but the selected policy is not necessarily explicit and clearly visible.

In the literature, the work on the has-part semantics has mainly concentrated on what is required of the individuals (Kim et al. 1987, Kim et al. 1989, Halper et al. 1992). For example, how many component individuals there can be in a particular part relation, whether a component individual may exist without being a part and whether the deletion of a component individual implies the deletion of the composite. The weak conformance of Study III takes a wider perspective to conformance and assumes conformance only for the creation time of the individual. In Study V, these semantics were addressed in detail, but conformance was only required for element structures, which are coarse descriptions of product individuals.

Object-Oriented Composition Modelling with Versions

One of the basic models for composite objects in engineering is that of ORION (Kim et al. 1987, Kim et al. 1989). In the model, the composition, i.e., the has-part relation, is modelled between classes. A class is considered as a collection of instances, which may either be generic instances or version instances. A has-part relation between instances may be from a version instance to a generic instance or version instance, i.e., types v-v and v-g relations discussed in this thesis. They consider the

evolution of schema to be covered by means of schema versioning (Kim and Chou 1988) and thus they do not combine class versioning with the has-part relation or discuss the propagation of changes in a class hierarchy.

Talens et al. (1993) have proposed a model, which incorporates an is-a hierarchy and composite objects. In their model, the has-part relation is given between classes and between versions; has-part relations between versions must follow the relations between classes. Their approach provides a powerful mechanism for explicitly controlling the versioning and version derivation of classes. The lack of generic references with a resolution mechanism (e.g., based on effectivity) means that the references are of type v-v. Thus, when a new version is created for a component, the version does not come into use without the creation of a new version for the composite or updating the has-part reference in the composite. That is, their approach focuses more on explicit version management, e.g., during design process, than on the product data management, e.g., support for "form, fit and function" principle. Moreover, they do not address the propagation of changes in an is-a hierarchy.

Andonoff et al. (Andonoff et al. 1995, Andonoff et al. 1996) carry on by taking into account the evolution of both the classes and individuals. They speak of version derivation hierarchies without explicitly representing generic objects. Therefore, they only have v-v type has-part and is-a relations, which they extend to contain combinations of versionable and non-versionable objects, either classes or individuals. With the is-a hierarchy, they maintain, however, that the versioning of superclass does not propagate to its subclasses, which is in contrast to what was discussed in this thesis.

Advanced Concepts in Modelling Product Families

The summary part of this thesis made a simplification by not discussing all the concepts needed in product family modelling. Product family modelling needs concepts for expressing conditions, such as components requiring other components, components being incompatible and so on (see, e.g. Peltonen et al. 1998, Soininen et al. 1998). Study II actually included a powerful mechanism for incorporating such conditions as properties of objects, which are inherited along is-a relations. An object's own conditions and the inherited ones are simply combined by logical ANDs. This summary part of the thesis concentrated on providing the

conceptual basis for a system that would integrate the evolution of product family schemas and individuals, and thus the details of modelling product families were intentionally kept to a minimum. The problem of conceptual mismatch was argued to be more relevant in modelling product families than in other kinds of products. Therefore, the evolution of schema and individuals in the form presented in this thesis is specifically a problem of product families.

Software Configuration Management and PDM

In software engineering, the software configuration management (SCM) is a close relative to PDM of mechanical products. There are certain differences between these fields but the gap between them is narrowing (Conradi and Westfechtel 1998, Estublier et al. 1998, Westfechtel and Conradi 1998). One particular difference is that, in SCM, product structure has not played such a central role as in PDM. In software, the product structure may be seen as a flat set of components (Reichenberger 1995) or it may reflect the directory hierarchy in which the source files are stored (Estublier et al. 1998), as it may be more important to express dependencies between objects (files, modules or syntactical object such as classes). However, in the future, software will also be a major player in traditionally mechanical products. Software modules will be components in product structures and much of the product variety will be created by means of software. It is likely that experiences in PDM will influence some parts of software engineering, e.g., within software configuration management, software architectures, software product lines, software generation and software product families.

Concept of Effectivity in Products

In the proposed framework, the effectivity was rather simple. That is, with a given time instant *t*, all objects would be bound to the version effective at that time, if any. For example, for a particular product one would get the product structure in which all versions are effective at *t*.

In practice, however, there are many time points, which can be considered as the start of the effectivity of a design object. At some time the design is approved and released for use. Shortly after that, the configuration models are updated and sales personnel (officially) start selling the new product. Purchasing of critical components may have begun much earlier and manufacturing will start using the new design gradually when the old work in progress is finished. Then finally, the first product

individual made according to the new design comes out of the manufacturing line. That is sometimes considered the start time of the effectivity of the new design because, from that point on, the new product individuals reflect the new design.

This all may require that, for example, in manufacturing multiple effectivities need to be recorded for a single design object. Separate effectivities may also be needed for different views, such as the sales configuration and manufacturing view of products. Between views, the semantics of effectivity may be quite different. Moreover, one object may need different effectivities for different market areas, if products are not launched and withdrawn globally at the same time. These examples show that the practical aspects of effectivity are very complicated. Since no definitive solution exists, the problem deserves further attention before information systems can capture all nuances of effectivity.

In this thesis, single effectivity was used. That is the effectivity controlling the configuration models. The creation time of a product individual, i.e., start of the effectivity of the first version, would be the time it was configured. If no changes are made to the individual during manufacturing, it will be delivered as configured. In practice, this need not be the case, however. Since the evolution of the individual is captured by its versions, the modifications in the manufacturing, i.e., even before the individual physically existed, should be recorded for the individual similarly to the way in which the after-sales modifications will be.

7 Conclusions

The results of this thesis can be summarised by the following few points. First, there was an observation of the conceptual mismatch raising the awareness of the problem and leading to requirements for different conceptual modelling methods from those directly provided by traditional databases or current PDM systems. The observation was backed up by a detailed study of product family modelling in the largest product data modelling standard, STEP. The study revealed the problems of the class instance paradigm when the schema is considered static, as is the case in traditional databases and certainly in standardisation.

In the search for a conceptual platform for an advanced PDM system, the importance of capturing the semantics of evolution was emphasised. The semantics was captured for modelling product families and their individuals by defining a set of invariants controlling the usage of concepts. In addition, dynamic referencing mechanisms were investigated in detail to allow more accurate selection of the needed semantics, including the support for the so-called "form, fit and function" versioning policy utilised in many companies.

The defined framework ties together the evolution of schema and individuals that only weakly conform to the schema. A study on the relation of schema and individuals of product families in the instantiation process, also called configuration process, was conducted. The study pointed out the unclear line between the schema and individuals as the configuration process gradually narrows the variety expressed in the schema to the representation of a single individual. The study also investigated the use of the uniform object paradigm for modelling product families, which was then taken as a basis for the conceptual platform.

The evolution of product individuals was acknowledged as a wide problem area and impossible to systematically control in its extreme, that is, when full freedom in modifying individuals is exercised. Two detailed studies, one on reconfiguration and the other on elaborate modelling of product individuals with multiple levels of abstraction, were conducted to address the evolution of product individuals. The approach for modelling reconfiguration knowledge did not include the whole history of the schema in one reconfiguration model. Instead, the approach relied on the way some investigated companies have developed for their after-sales support, that is, to define reconfiguration operations that can be

applied to product individuals under given preconditions. The other detailed study provided a methodology for modelling product individuals in multiple abstraction levels. The study was based on the experiences with a company manufacturing configurable products that have a long individual lifetime. The long lifetime and large variety of product individuals make systematic management of individuals difficult. After-sales is, however, a remarkable business for such companies, typically totalling close to or more than half of all sales.

On the whole, the results of this thesis provide a conceptual basis for capturing the evolution of product families and their individuals in the scope set for an advanced PDM systems in Figure 1. The idea is that an advanced PDM system would have the capability of understanding the semantics so that the product data management policies of the company could be encoded in the system, which would then control the selected policies.

8 Kiitokset / Acknowledgements

Tämä väitöskirjatyö on tehty Teknillisen korkeakoulun TAI-tutkimuslaitoksen tuotetiedonhallinnan tutkimusryhmässä (PDMG). Haluan kiittää työn onnistumisesta kaikkia PDMG:n nykyisiä ja entisiä jäseniä, jotka ovat omalla panoksellaan luoneet tutkimustyölle sopivan, vireän ja miellyttävän työyhteisön.

Suurin kiitos näiden työskentelymahdollisuuksien luomisesta kuuluu *professori Shosta Suloselle,* joka on pitkäjänteisesti ja määrätietoisesti rakentanut ryhmämme identiteettiä useiden vuosien ja projektien yli. Haluan erityisesti kiittää Sinua siitä henkilökohtaisesta tavasta, jolla olet minuun ja työhöni suhtautunut. Hyvin tärkeitä ovat olleet myös ne työhön kuulumattomat asiat, jotka olen Sinulta oppinut ja joita olemme yhdessä kokeneet.

TkT Hannu Peltosta haluan kiittää koko tutkijaurani jatkuneesta hyvästä yhteistyöstä. Olen saanut nauttia laajasta asiantuntemuksestasi sekä lahjomattoman terävästä ja kriittisestä otteestasi niin käsitteisiin kuin termeihin, joita olemme vuosien varrella yhdessä pyöritelleet.

TkL Juha Tiihonen on tehnyt ryhmässä merkittävää työtä tuotteiden konfiguroinnin alueella ja samalla kartoittanut pohjaa sille käytännön ilmiölle, jonka päällä tämä väitöskirjatyö seisoo. Sinulle kuuluu myös erityiskiitos työn ulkopuolisista hetkistä valokuvauksen parissa.

TkL Timo Soininen on ansiokkaasti opastanut ryhmäämme formaalien menetelmien käytössä, mutta myös kiitettävästi soveltanut epäformaaleja menetelmiä ryhmän hengen luomiseen. Haluan kiittää Sinua asiantuntemuksestasi tietämyksen ja logiikan alalta, jota olet tämän työn kuluessa avukseni tarjonnut; kuten myös niistä filosofisista pohdinnoista, jotka ovat toisinaan johtaneet…, no ties minne, mutta silti aina mielenkiintoisille alueille.

Tutkimusjohtaja Asko Martio on tuonut ryhmäämme teollista kokemusta aiemmalta työuraltaan. Haluan kiittää Sinua kokemuksesta tulleista kommenteistasi, jotka ovat tehokkaasti maadoittaneet tämänkin väitöskirjatyön ympärillä keveimmin leijuneita ajatuksia. Mukavia ovat olleet myös pitkät automatkat kanssasi läntisen Suomen yhteistyöyrityksiin.

Kiitän myös *TkL Kari Alhoa* tutkijaurani alkuaikojen yhteistyöstä, joka heijastuu yhdessä tämän väitöskirjan osatyössä.

Työtä ovat rahoittaneet Tekes, Suomen Akatemia ja Teknillisen korkeakoulun tukisäätiö. Olen myös ollut opiskelijana HeCSE (Helsinki Graduate School of Computer Science and Engineering) -tutkijakou-

lussa, jonka kautta olen saanut rahallista ja henkistä tukea tohtoriopinnoilleni. Haluankin kiittää *professori Martti Mäntylää* myönteisestä ja kannustavasta asenteesta, jolla hän on HeCSEn johtajana seurannut väitöskirjatyöni etenemistä.

Työni esitarkastajia *professori Pasi Tyrväistä* ja *TkT Jouko Vuoskoskea* kiitän arvokkaista kommenteista, joiden toteuttaminen konkretisoi ja selkeytti työtä.

I express my kind thanks to Ms. Ruth Vilmi for reviewing the English language of this thesis.

Lopuksi kuuluvat kiitokset puolisolleni *ETT Satu Männistölle.* Esimerkilläsi avasit latua väitöskirjan tekemiseen, mutta ennen muuta tarjosit sen henkisen nojan, jota tällaisen prosessin aikana tarvitsee. Varsinainen kiitos Sinulle, Satu, kuitenkin kuuluu pitkäaikaisesta läheisyydestä ja ystävyydestä, jossa olen saanut kanssasi kasvaa.

Helsingissä 8.6.2000

Tomi Männistö

References

Abiteboul S, Hull R. IFO: A formal semantic database model. ACM Transactions on Database Systems 1987;12(4):525–65.

Ahmed R, Navathe SB. Version management of composite objects in CAD databases. Proc. of the International Conference on Management of Data (SIGMOD); ACM; 1991. p. 218–27.

Allen JF. Maintaining knowledge about temporal intervals. Communications of the ACM 1983;26(11):832–43.

Almarode J. Rule-based delegation for prototypes. Conference on Object-Oriented Programming Systems and Languages (OOPSLA); 1989. p. 363–70.

Andonoff E, Hubert G, Le Parc A, Zurfluh G. Modelling inheritance, composition and relationship links between object versions and class versions. Advances Information Systems Engineering. Proceedings of the 7th International Conference (CAiSE'95); 1995. p. 96–111.

Andonoff E, Hubert G, Le Parc A, Zurfluh G. Integrating versions in the OMT models. Proc. of the 15th International Conference on Conceptual Modeling (ER'96); Springer-Verlag; 1996. p. 427–87.

Artale A, Franconi E, Guarino N, Pazzi L. Part-whole relations in object-centered systems: An overview. Data & Knowledge Engineering 1996;3(20):347–83.

Banerjee J, Chou H-T, Garza JF, Kim W, Woelk D, Ballou N, Kim H-J. Data model issues for object-oriented applications. ACM Transactions on Office Information Systems 1987;5(1):3–26.

Banerjee J, Kim W, Kim H-J, Korth HF. Semantics and implementation of schema evolution in object-oriented databases. Proc. of the international conference on management of data (SIGMOD); 1987. p. 311–22.

Barker VE, O'Connor DE. Expert systems for configuration at Digital: XCON and beyond. Communications of the ACM 1989;32(3):298–318.

Batini C, Ceri S, Navathe SB. Conceptual database design. The Benjamin/Cummings; 1992.

Batory DS, Kim W. Modeling concepts for VLSI CAD objects. ACM Transactions on Database Systems 1985;10(3):322–46.

Bettini C, Dyreson CE, vans WS, Snodgrass RT, Wang XS. Glossary of time granularity concepts. In: Etzion O, Jajodia S, Spirada S, editors. Temporal Databases — Research and Practice. Springer-Verlag; 1998. p. 406–13.

Biliris A. Database support for evolving design objects. Proc. of the 26th ACM/IEEE design automation conference; 1989. p. 258–63.

Björnerstedt A, Hultén C. Version control in an object-oriented architecture. In: Kim W, Lochovsky FH, editors. Object-oriented concepts, databases, and applications. Addison-Wesley; 1989. p. 451–85.

Casais E. Managing class evolution in object-oriented systems. In: Tsichritzis D, editor. Object management. Centre Universitaire d'Informatique, University of Geneve; 1990. p. 133–95.

Chou HT, Kim W. A unifying framework for version control in a CAD environment. Proc. of the 12th international conference on very large databases (VLDB); 1986. p. 336–44.

Clamen SM. Type evolution and instance adaptation. Technical report. School of Computer Science, Carnegie Mellon University; 1992. Report No.: CMU-CS-92–133.

Conradi R, Westfechtel B. Version models for software configuration management. ACM Computing Surveys 1998;30(2):232–82.

Cunis R, Günter A, Syska I, Peters H, Bode H. PLAKON — An approach to domain-independent construction. The second international conference on industrial & engineering applications of artificial intelligence & expert systems IEA/AIE-89; 1989. p. 866–74.

Darr T, McGuinness D, Klein M, editors. Special Issue on Configuration Design. AI EDAM 1998;12(4).

De Castro C, Grandi F, Scalas MR. Schema versioning for multitemporal relational databases. Information Systems 1997;22(5):249–90.

Demaid A, Zucker J. Prototype-oriented representation of engineering design knowledge. Artificial Intelligence in Engineering 1992;7(1):47–61.

Dittrich KR, Lorie RA. Version support for engineering database systems. IEEE Transactions on Software Engineering 1988;14(4):429–36.

Eastman CM, Fereshetian N. Information models for use in product design: A comparison. Computer-Aided Design 1994;26(7):551–72.

Erens F. The synthesis of variety—Developing product families [PhD thesis]. Eindhoven University of Technology; 1996.

Erens F, McKay A, Bloor S. Product modelling using multiple levels of abstraction. Instances as types. Computers in Industry 1994;24(1):17–28.

Estublier J, Casallas R. Three dimensional versioning. In: Estublier J, editor. Software Configuration Management: ICSE SCM-4 and SCM-5 Workshops: Selected papers. Lecture Notes in Computer Science, vol. 1005; Springer-Verlag; 1995. p. 118–35.

Estublier J, Favre J-M, Morat P. Towards SCM/PDM integration? In: Magnusson B, editor. Proc. of European Conference in Object-Oriented Programming (ECOOP 98), Software Configuration Management SCM-8, LNCS 1439; Springer-Verlag; 1998. p. 75–94.

Faltings B, Freuder EC, editors. Configuration—Papers from the 1996 AAAI fall symposium. (AAAI Technical report FS-96–03) AAAI Press; 1996.

Faltings B, Freuder EC, editors. Special Issue on Configuration. IEEE intelligent systems & their applications 1998;13(4):29–85.

Faltings B, Freuder EC, Friedrich GE, Felfernig A, editors. Configuration—Papers from the 1999 AAAI workshop. (AAAI Technical report FS-99–05.) AAAI Press; 1999.

Feiler PH. Configuration management models in commercial environments. Software Engineering Institute, Carnegie-Mellon University; 1991. Report No.: CMU/SEI-91-TR-7.

Fleishandler G, Friedrich GE, Haselböck A, Schreiner H, Stumptner M. Configuring large systems using generative constraint satisfaction. IEEE intelligent systems & their applications 1998;13(4):59–68.

Fowler M. UML distilled: Applying the standard object modeling language. Addison-Wesley; 1997.

Gerstl P, Pribbenow S. Midwinters, end games and body parts—A classification of part-whole relations. International Journal of Human-Computer Studies 1995;43(5/6):865–89.

Gibbs S, Tsichritzis D, Casais E, Nierstrasz O, Pintado X. Class management for software communities. Communications of the ACM 1990;33(9):90–103.

Goralwalla I, Özsu MT, Szafron D. An object-oriented framework for temporal data models. In: Etzion O, Jajodia S, Sripada S, editors. Temporal Databases—Research and Practice. Springer-Verlag; 1998. p. 1–35.

Goyal P, Qu Y-Z, Sadri F. The temporal object model. In: Srinivasan B, Zeleznikow J, editors. Proc. of the 3rd Australian Database Conference; 1992. p. 36–50.

Hakim M. Modeling evolving information about engineering design products [PhD thesis]. Department of Civil Engineering, Carnegie Mellon University; 1993.

Hakim MM, Garrett JH. An object-centered approach for modelling engineering design products: Combining description logic and object-oriented modelling. AI EDAM 1997;11(3):187–98.

Halper M, Geller J, Perl Y. An OODB "Part" relationship model. Proc. of the information and knowledge management (CIKM-92); 1992. p. 602–11.

Hayes-Roth F. Rule-based systems. Communications of the ACM 1985;28(9):921–32.

Heinrich M, Jüngst W. A resource-based paradigm for the configuring of technical systems from modular components. Proc. of the seventh IEEE conference on artificial intelligence applications; IEEE; 1991. p. 257–64.

Heinrich M, Jüngst W. The resource-based paradigm: Configuring technical systems from modular components. In: Faltings B, Freuder EC, editors. Configuration—papers from the 1996 AAAI Fall Symposium. AAAI Press; 1996. p. 19–27.

Ho T-H, Tang CS, editors. Product variety management: Research advances. Kluwer Academic Publishers; 1998.

ISO. ISO Standard 10303–1: Industrial automation systems and integration — Product data representation and exchange — Part 1: Overview and fundamental principles. 1994a.

ISO. ISO Standard 10303–11: Industrial automation systems and integration — Product data representation and exchange — Part 11: Description methods: The EXPRESS language reference manual. 1994b.

ISO. ISO Standard 10303–203: Industrial automation systems and integration — Product data representation and exchange — Part 203: Application protocol: Configuration controlled design. 1994c.

ISO. ISO Standard 10303–41: Industrial automation systems and integration — Product data representation and exchange — Part 41: Integrated generic resources: Fundamentals of product description and support. 1994d.

ISO. ISO Standard 10303–44: Industrial automation systems and integration — Product data representation and exchange — Part 44: Integrated generic resources: Product structure configuration. 1994e.

ISO. ISO Committee Draft 10303–214: Industrial automation systems and integration — Product data representation and exchange — Part 214: Application protocol: Core data for automotive mechanical design process. 1995a.

ISO. ISO Draft International Standard 13584–10: Industrial automation systems and integration — Parts Library — Part 10: Conceptual model of parts library. 1995b.

ISO. ISO Draft International Standard 13584–42: Industrial automation systems and integration — Parts Library — Part 42: Methodology for structuring part families. 1996.

ISO. Guide on STEPlib, ISO TC184/SC4/WG3 N424. 1997a.

ISO. ISO Committee Draft 10303–221: Industrial automation systems and integration — Product data representation and exchange — Part 221: Application Protocol: Functional data and their schematic representation for process plant. 1997b.

ISO. ISO Draft International Standard 13584–1: Industrial automation systems and integration — Parts Library — Part 1: Overview and fundamental principles. 1997c.

ISO. ISO Second Committee Draft 10303–214: Industrial automation systems and integration — Product data representation and exchange — Part 214: Application protocol: Core data for automotive mechanical design process. 1997d.

ISO. ISO Working draft: Product Data Representation and Exchange, ISO TC184/SC4/WG11 N088. 1999.

Jensen CS, Snodgrass RT. Semantics of time-varying information. Information Systems 1996;21(4):311–52.

Jensen CS, Dyreson CE, Böhlen M, Clifford J, Elmasri R, Gadia SK, Grandi F, Hayes P, Jajodia S, Käfer W, Kline N, Lorenzos N, Mitsopoulos Y, Montanari A, Nonen D, Peressi E, Pernici B, Roddick JF, Sarda NL, Scalas MR, Segev A, Snodgrass RT, Soo MD, Tansel A, Tiberio P, Wiederhold G. The concensus glossary of termporal database concepts — February 1998 version. In: Etzion O, Jajodia S, Spirada S, editors. Temporal Databases — Research and Practice. Springer-Verlag; 1998. p. 367–405.

Jørgensen KA. Object-oriented information modelling. In: Modern manufacturing—Information control and technology. Springer-Verlag; 1994. p. 47–84.

Jørgensen KA, Raunsbaek T. Design of product configuration management systems. Proceedings of 2nd International Conference on Engineering Design and Automation. Integrated Technology Systems, Inc.; 1998.

Joseph J, Shadowens M, Chen J, Thompson G. Strawman reference model for change management of objects. Computer Standards & Interfaces 1991:249–69.

Jüngst W, Heinrich M. Using resource balancing to configure modular systems. IEEE intelligent systems & their applications 1998;13(4):50–8.

Katz RH. Toward a unified framework for version modeling in engineering databases. ACM Computing Surveys 1990;22(4):375–408.

Katz RH, Chang E. Managing change in a computer-aided design database. Proc. of the 13th international conference on very large databases (VLDB); 1987. p. 455–62.

Katz RH, Chang E, Bhateja R. Version modeling concepts for computer-aided design databases. Proc. of the international conference on management of data (SIGMOD); 1986. p. 379–86.

Kemper A, Moerkotte G. Object-oriented database management. Applications in engineering and computer science. Prentice Hall; 1994.

Kim W, Banerjee J, Chou HT. Composite object support in an object-oriented database system. Proc. of conference on object-oriented programming systems and languages (OOPSLA); 1987. p. 118–25.

Kim W, Bertino E, Garza JF. Composite objects revisited. Proc. of the international conference on management of data (SIGMOD); 1989. p. 337–47.

Kim W, Chou HT. Versions of schema for object-oriented databases. Proc. of the 14th international conference on very large databases (VLDB); 1988. p. 148–59.

Lieberman H. Using prototype objects to implement shared behavior in object oriented systems. Conference on object-oriented programming systems and languages (OOPSLA); 1986. p. 214–23.

Männistö T. Towards management of evolution in product configuration data models [Licentiate thesis]. Laboratory of Information Processing Science, Department of Computer Science, Helsinki University of Technology; 1998.

Männistö T, Peltonen H, Alho K, Sulonen R. A framework for long term information management of product configurations. Laboratory of Information Processing Science, Helsinki University of Technology; 1993. Report No.: TKO-B105.

Männistö T, Peltonen H, Sulonen R. Open data modelling issues in product configuration. Laboratory of Information Processing Science, Helsinki University of Technology; 1995. Report No.: TKO-B127.

Männistö T, Peltonen H, Sulonen R. View to product configuration knowledge modelling and evolution. In: Faltings B, Freuder EC, editors. Configuration—papers from the 1996 AAAI Fall Symposium (AAAI technical report FS-96–03). AAAI Press; 1996. p. 111–8.

McGuinness D, Wright JR. An industrial-strength description-logics-based configurator platform. IEEE intelligent systems & their applications 1998a;13(4):69–77.

McGuinness D, Wright JR. Conceptual modelling for configuration: A description logic-based approach. AI EDAM 1998b;12(4):333–44.

McKay A, Erens F, Bloor MS. Relating Product Definition and Product Variety. Research in Engineering Design 1996;8(2):63–80.

Miller E, MacKrell J, Mendel A. PDM buyer´s guide. (6th edition) CIMdata Corporation; 1997.

Mittal S, Araya A. A knowledge-based framework for design. Artificial intelligence in engineering design. Academic Press Inc.; 1992. p. 273–93.

Mittal S, Frayman F. Towards a generic model of configuration tasks. Proc. of the 11th International Joint Conference on Artificial Intelligence (IJCAI); 1989. p. 1395–401.

Monk S, Sommerville I. Schema evolution in OODBs using class versioning. SIGMOD Record 1993;22(3):16–22.

Mülle JA, Dittrich KR, Kolz AM. Design management support by advanced database facilities. In: Rammig FJ, editor. IFIP Workshop on Tool Integration and Design Environment; Elesevier Science Publishers B.V. (North-Holland); 1988. p. 23–49.

Mylopoulos J. Information modeling in the time of revolution. Information Systems 1998;23(3/4):127–55.

Nguyen GT, Rieu D. Schema change propagation in object-oriented databases. Information Processing 89. Proceedings of the IFIP 11th World Computer Gongress.; North-Holland; 1989a. p. 815–20.

Nguyen GT, Rieu D. Schema evolution in object-oriented database systems. Data & Knowledge Engineering 1989b;4(1):43–67.

Nierstrasz O. A survey of object-oriented concepts. In: Kim W, Lochovsky FH, editors. Object-Oriented Concepts, Databases and Issues. Addison-Wesley; 1989. p. 3–21.

Owen J. STEP—An introduction. (2nd edition) Information Geometers; 1997.

Parsons J, Wand Y. Choosing classes in conceptual modeling. Communications of the ACM 1997;40(6):63–9.

Peltonen H. Concepts and an Implementation for Product Data Management [PhD thesis]. Helsinki University of Technology, Department of Computer Science and Engineering; 2000.

Peltonen H, Männistö T, Soininen T, Tiihonen J, Martio A, Sulonen R. Concepts for modelling configurable products. Proc. of the Product Data Technology Days; Quality Marketing Services; 1998. p. 189–96.

Penney J, Stein J. Class modification in the GemStone object-oriented DBMS. Conference on object-oriented programming systems and languages (OOP-SLA); 1987. p. 111–7.

Peters RJ. TIGUKAT: A uniform behavioral objectbase management system [PhD thesis]. Department of Computing Science, The University of Alberta; 1994.

Peters RJ, Özsu MT. Reflections in uniform behavioral object model. Proc. of the 12th international comference on entity-relationship approach (ERA'93); 1993. p. 37–49.

Peters RJ, Özsu MT. An axiomatization model of dynamic schema evolution in objectbase systems. ACM Transactions on Database Systems 1997;22(1):75–114.

Ramakrishnan R, Ram DJ. Modeling design versions. In: Vijayaraman TM, Buchmann AP, Mohan C, Sarda NL, editors. Proc. of the 22nd international conference on very large databases (VLDB); 1996.

Reichenberger C. Orthogonal version management. International workshop on software configuration management. Software engineering notes; ACM; 1989. p. 137–40.

Reichenberger C. VOODOO A Tool for orthogonal version management. In: Estublier J, editor. Software Configuration Management: ICSE SCM-4 and SCM-5 Workshops: Selected papers. Lecture Notes in Computer Science, vol. 1005; Springer-Verlag; 1995. p. 61–79.

Roddick JF. Schema evolution in database systems—An annotated bibliography. SIGMOD Record 1992;21(4):35–40.

Roddick JF. A survey of schema versioning issues for database systems. Information and Software Technology 1995;37(7):383–93.

Roddick JF. A model for schema versioning in temporal database systems. Australian Computer Science Communications 1996;18(1):446–52.

Rumbaugh JE, Blaha MR, Premerlani WJ, Eddy F, Lorensen W. Object-oriented modeling and design. Prentice-Hall; 1991.

Sabin D, Freuder EC. Configuration as composite constraint satisfaction. In: Faltings B, Freuder EC, editors. Configuration—papers from the 1996 AAAI Fall Symposium. AAAI Press; 1996. p. 28–36.

Sanderson SW, Uzumeri M. The innovation imperative: Strategies for managing product models and families. Irwin Professional Publishing; 1997.

SAP. Der SAP Konfigurator (reference manual). 1994.

Schönsleben P, Oldenkott H. Enlarging CAD and interfaces between PPC and CAD to respond to product configuration requirements. In: Pels HJ, Wortmann JC, editors. Integration in production management systems. Elsevier Science Publishers B.V.; 1992. p. 53–69.

Searls DB, Norton LM. Logic-based configuration with a semantic network. Journal of Logic Programming 1990:53–73.

Simon HA. The sciences of the artificial. (3rd edition) MIT Press; 1996.

Simons P. Parts — A study in ontology. Clarendon Press; 1987.

Skarra AH, Zdonik SB. The management of changing types in an object-oriented database. Conference on object-oriented programming systems and languages (OOPSLA); 1986. p. 483–91.

Skarra AH, Zdonik SB. Type evolution in an object-oriented database. In: Shiver B, Wegner P, editors. Directions in object-oriented programming. MIT Press; 1988. p. 393–415.

Snodgrass RT. Developing time-oriented database applications in SQL. Morgan Kaufmann; 2000.

Soininen T. Product configuration knowledge: Case study and general model [Master's thesis]. Helsinki University of Technology; 1996.

Soininen T. An approach to configuration knowledge representation and reasoning [Licentiate thesis]. Laboratory of Information Processing Science, Department of Computer Science, Helsinki University of Technology; 1998.

Soininen T, Gelle E. Dynamic constraint satisfaction in configuration. In: Faltings B, Freuder EC, Friedrich GE, Felfernig A, editors. Configuration—Papers from the AAAI Workshop; AAAI; 1999. p. 95–106.

Soininen T, Tiihonen J, Männistö T, Sulonen R. Towards a general óntology of configuration. AI EDAM 1998;12(4):357–72.

Stefik MJ, Bobrow DG. Object-oriented programming: Themes and variations. AI Magazine 1986;6(4):40–62.

Stein LA. Delegation is inheritance. Conference on object-oriented programming systems and languages (OOPSLA); 1987. p. 138–46.

Stumptner M, Wotawa F. Model-based reconfiguration. Proceedings of the 5th conference on Artificial Intelligence in Design '98; 1998.

Su SYW, Hyun SJ, Chen H-HM. Temporal association algebra: A mathematical foundation for processing object-oriented temportal databases. IEEE Transactions on Knowledge and Data Engineering 1998;10(3):389–407.

Taivalsaari A. Classes versus prototypes: Some philosophical and historical observations. Journal of Object-Oriented Programming 1997;(November / December):44–50.

Talens G, Oussalah C, Colinas MF. Versions of simple and composite objects. Proc. of the 19th VLDB Conference; Morgan Kaufman; 1993. p. 62–72.

Tan L, Katayama T. Meta operations for type management in object-oriented databases. In: Kim W, Nicolas JM, Nishio S, editors. Deductive and object-oriented databases (DOOD89); Elsevier Science Publishers B.V.; 1989. p. 241–58.

Tanaka K, Yosikawa, Ishihara. Schema design, views and incomplete information in object-oriented databases. Journal of Information Processing 1989;12(3):239–50.

Theodoulidis CI, Loucopoulos P. The time dimension in conceptual modelling. Information Systems 1991;16(3):273–300.

Tiihonen J. Computer-assisted elevator configuration [Master's thesis]. Helsinki University of Technology; 1994.

Tiihonen J. National product configuration survey — Customer specific adaptation in the Finnish industry (in Finnish) [Licentiate thesis]. Laboratory of Information Processing Science, Department of Computer Science, Helsinki University of Technology; 1999.

Tiihonen J, Männistö T, Peltonen H, Sulonen R. Selection tables in engineering design. KARPS-95 Second International Symposium on Knowledge Acquisition, Representation and Processing; 1995. p. 118–20.

Tiihonen J, Soininen T, Männistö T, Sulonen R. State-of-the-practice in product configuration—A survey of 10 cases in the Finnish industry. In: Tomiyama T, Mäntylä M, Finger S, editors. Knowledge intensive CAD. London: Chapman & Hall; 1996. p. 95–114.

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

Törmä S. A model for the dynamic planning of industrial projects [PhD thesis]. Helsinki University of Technology, Department of Computer Science and Engineering; 1997.

Ullman J. Principles of database and knowledgebase systems. Volume I. Computer Science Press, Inc.; 1988.

van den Hamer P, Lepoeter K. Managing design data: The five dimensions of CAD frameworks, configuration management, and product data management. Proceedings of the IEEE 1996;84(1):42–56.

van Veen EA. Modelling product structures by generic Bills-of-Material [PhD thesis]. Eindhoven University of Technology; 1991.

Wagner FR, Lima AHV. Design version management in the GARDEN framework. Proc. of the 28th ACM/IEEE design automation conference; 1991. p. 704–10.

Wedekind H. Are terms "version" and "variant" orthogonal to one another? A critical assessment of the STEP standardization. SIGMOD Record 1994;23(4):3–7.

Wegner P. Concepts and paradigms of object-oriented programming. OOPS Messenger 1990;1(1):7–87.

Weida R. Closed terminologies in description logics. In: Faltings B, Freuder EC, editors. Configuration—Papers from the 1996 AAAI Fall Symposium. AAAI Press; 1996. p. 11–8.

Weigel R, Faltings B. Abstraction techniques for configuration systems. In: Faltings B, Freuder EC, editors. Configuration—Papers from the 1996 AAAI Fall Symposium. AAAI Press; 1996. p. 55–60.

Westfechtel B, Conradi R. Software configuration management and engineering data management: Differences and similarities. In: Magnusson B, editor. Proc. of European Conference in Object-Oriented Programming (ECOOP 98), Software Configuration Management SCM-8, LNCS 1439; Springer-Verlag; 1998. p. 95–106.

Winston ME, Chaffin R, Herrman D. A taxonomy of part-whole relations. Cognitive Science 1987;11(4):417–44.

Zdonik SB, Maier D. Fundamentals of object-oriented databases. Readings in object-oriented databases. Morgan Kaufmann; 1990. p. 1–32.

Zicari R. A framework for schema updates in an object-oriented database system. Proc. seventh international conference on data engineering; 1991. p. 2–13.

Zicari R. A framework for schema updates in an object-oriented database system. In: Bancilhon F, Delobel C, Kanellakis P, editors. Building an object-oriented database system. The story of O_2. Morgan Kaufmann Publishers, Inc.; 1992. p. 256–77.

Zucker J, Demaid A. Modelling heterogenous engineering knowledge as transactions between delegating objects. Proc. 2nd international conference on artificial intelligence in design; 1992a.

Zucker J, Demaid A. The rôle of evolutionary inheritance in developing knowledge bases for conceptual design of engineering products. European Journal of Engineering Education 1992b;17(2):173–80.