A Framework for Long Term Information Management of Product Configurations

Tomi Männistö, Hannu Peltonen, Kari Alho, and Reijo Sulonen
Helsinki University of Technology
Department of Computer Science
Otakaari 1, 02150 Espoo, Finland
E-mail: tmm@cs.hut.fi

Abstract

Product configuration is an information intensive task for creating valid component combinations or descriptions from specifications. Information included in this process consists of product models, component models, actual partially or fully specified configurations, plus information of the configuration process itself. A system utilizing only a snapshot of this information can be built in a straightforward manner, but keeping all the information up-to-date within a constantly evolving environment is far from trivial. So far there has not been much research done on how this evolution of knowledge could be mastered. We start by describing the requirements for a configuration management system for a world-wide industrial enterprise. We then present a data model for a configuration environment where multiple partners take part in the process. From data engineering point of view the main problem is that configuration data in the system should be accessible for long periods of time and the system should be able to cope with information changes in all categories described above.

1 Introduction

The management of engineering data has been actively researched. Various ideas have been presented in product modelling, versioning, part-of semantics, configuration management, etc. (see [8]). These solutions have given guidelines for the evolution of the engineering data modelling. However, many of the presented approaches need further research before they can be applied for current problems of industry. We start from a configuration data management problem of a lift manufacturing company, try to abstract it, and attempt to find a solution using more general concepts.

We do not intend to solve a general configuration problem since we strongly believe that there is no single problem. We feel that lifts are a good domain for configuration management research since the products vary from standard to semi- and fully custom-

ized. A model that defines the entities, operations, and processes for a lift configurator should also be applicable to a wide variety of other products.

The complexity of the configuration problem depends much on the degree of customization of the products. If only a predefined set of standard products is available for a customer, the configuration process reduces to the generation of a bill-of-materials, which can be done automatically from the customer specification. Main problems that remain are the modelling of the standard products and their options for a configuration system, and, more importantly, the maintenance of that knowledge.

Allowing more flexibility in customer specification leads directly to the possibility of iteration in the configuration process. Configuration process involves multiple partners who have to reach and maintain consensus of the common goal, the configuration. For example, if standard components are combined in a non-standard way, the product does not cause any configuration problems in the component manufacturing organizations. However, to reach the consensus it may be necessary to negotiate with the organizations responsible for designing or putting together the assembly. The situation is even more complex if the customer needs are fulfilled only with non-standard components, that have to be designed in component factories, which then become potential sources for disturbance in the process. To summarize, increasing customization implies increasing number of partners which have to be in agreement of the configuration. If anything changes, i.e., someone later disagrees, there has to be a mechanism for reaching consensus again.

The nature of a product's life-cycle also has a large impact on product modelling and configuration. The design, production, assembly, and maintenance of the product may be strictly ordered; that is, first design the product, then manufacture and assemble one or more pieces, and then maintain them according to the documentation created during or after the design. In this case the design is an intensive process with frequent changes, and concurrent engineering can be utilized at the CAD tool level. But when the production starts there should be no more changes to the design, since those may propagate to other products or components and might turn out very expensive.

In the other case, design, production, and maintenance take place more in parallel. This is the situation when it is not possible to design a small set of standard products and customize them using predefined options. Instead one designs a *product model*, which has some of its properties or components fixed or restricted to certain choices and some left unrestricted. Product models make it possible to reuse the effort that went into the preliminary design and still fulfil varying customer needs. The product models themselves are not static, but may evolve even after the manufacturing of products based on them has started. There will be new customer requirements that could not have been predicted, new regulations, new versions of components, etc. All these changes may have to be propagated to the product models.

A substantial amount of research has been carried out for configuring lifts automatically [7], [10], [16]. These approaches mainly aim at a fully automatic lift configurator where all product configuration knowledge is predefined. On the other hand, many such automatic configurators do not have a clear mechanism for knowledge base changes, so that it is up to the knowledge base designers to take into account all the consequences of

changing some rules. Some otherwise successful product configurators need a large amount of human resources for maintenance (e.g., Digital's XCON [1], [11]). We believe this problem can be reduced by dividing product configuration knowledge into more independent components, structuring it with specialization hierarchies, and making the knowledge less hard-wired by using a general reference mechanism between components of a composite object.

The product configuration process takes place in various steps during a products lifetime. Since the process may span different parties in different organizations, we need to apply process modelling concepts to be able to better understand, coordinate and automate these processes (see [4]).

This research is being done in cooperation with KONE Elevators, a Finnish company, which is the third largest lift manufacturer in the world with annual turnover more than US$2B (1992) and 15 000 new installations annually.

This paper is organized as follows. In section 2 we describe the configuration process and requirements for a configuration system. Section 3 gives an outline of configuration data as a specialization hierarchy and introduces our prototype model and tree transformation operations as a basis for a configuration system. Section 4 discusses briefly the future work of integrating the configurator with other systems.

2 Product Configuration Environment

2.1 Configuration Process Partners

The lift configuration process takes place in a setting illustrated in Figure 1. In the figure, letter S stands for "Sales", L for "Logistics centre", and C for "Component factory". The customer order arrives at the sales organization, which tries to translate the customer needs into a product specification. This specification is then forwarded to the logistics centre, which configures the product according to the specification, generating a number of component orders. Component factories manufacture the components, which are then delivered back to the logistics centre for shipping to the installation site.

If the customer orders a standard product, the configuration can be created entirely in the sales organization. The logistics centre only has to generate the component orders automatically.

When a customer wants something that is not a standard product, but can be manufactured using standard components, the main responsibility for the configuration process moves to the logistics centre. Moreover, if there is a need for a non-standard component that has to be designed in a component factory, the configuration decisions are no longer made only in sales and logistics centre, but the component factories are also involved in the process as active partners. The depth of the configuration process thus depends on the degree of the customization needed.

We can expect changes or disturbances in the configuration process from all active partners. A standard product configuration is likely to receive changes only from cus-

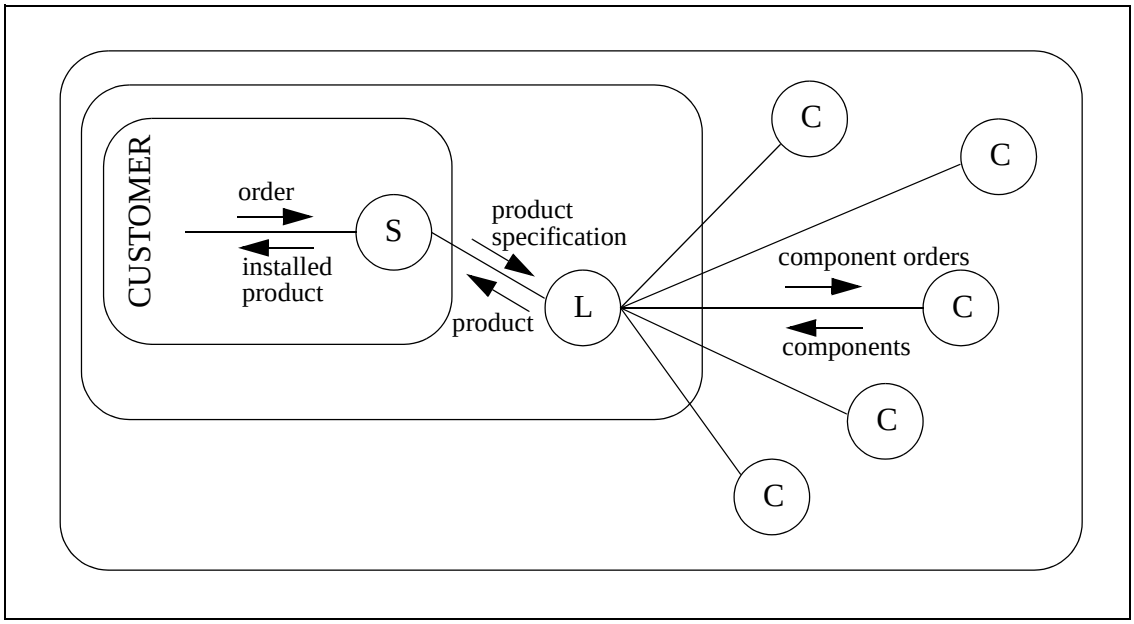

Figure 1: Configuration process

tomer or error fixes from the sales organization, whereas configurations with non-standard components are also subject to changes coming from the component factories.

Any change to the configuration possibly violates the consensus among the partners, which cannot proceed their tasks without reacting to the change. We are working on a workflow model that would include a protocol for reaching a consensus after a disagreement.

These problems go beyond organizational boundaries, which suggests that finding a consensus is a non-trivial task in practice. Since the partners in the configuration process are quite independent, there is no authority responsible for telling everyone what to do—instead, the partners have to find the consensus all by themselves.

2.2 Product Models

Some product models allow a configuration to be created automatically from the customer specification. These product models are totally pre-engineered and give restrictions to certain component combinations and attribute values.

Sometimes, however, the customer specification cannot be satisfied by any of these models. There are several ways of creating such non-standard configurations: Starting from a clean table and selecting the components on the basis of designer's experience, taking an old product as a starting point and making the needed component changes, or starting from an existing product model and then at a certain point of time departing the design from the restrictions of that product model.

In the first case, the configuration must still obey some general rules of the application domain; the configuration process thus never starts from a completely clean table. In the second and the third case, the constraints of the starting point (whether an old product or an existing product model) should be copied to the new design. Then the designer must

only make those modifications to the constraints that are absolutely necessary, reusing most of the existing knowledge.

2.3 Validity of a Configuration

The configurator should ultimately only produce configurations which satisfy certain conditions. These conditions are expressed as *constraints*. We define that a configuration is *valid* if none of its constraints are violated.

Constraints in a product model are declarative, i.e., they do not specify how they can be made valid. This information is given by *procedures*. The procedures can be classified into two categories: Generation and fixing procedures. *Generation procedures* are used for determining new values for unspecified components or attributes. *Fixing procedures*, on the other hand, try to bring an invalid configuration "closer" to a valid state. In many cases these procedures are attached to a single constraint for making it valid. The idea of fixes is presented in [10].

In addition to creating a valid configuration, the procedures often also try to optimize the result according to some criteria, such as cost. These procedures may, for example, find the least expensive engine by trying different engines in increasing order until the configuration becomes valid. To find the global optimum, several fixing procedures must be taken into account simultaneously [10].

At any moment during the configuration process only a subset of the available procedures are eligible for execution. The next procedure to be executed is selected using a selection algorithm. Procedures can have some priority information attached to them so that the procedure selection algorithm is able to sort the procedures according to the preferred execution order. At this point the algorithm can either automatically execute the best procedure—provided that an unambiguously best procedure exists—or ask the user to make the final selection. We do not want to specify any single selection algorithm, especially since in many cases user interaction may be the only way to make the decision, sorting being only an aid to the user.

At any time during the configuration process, the user should also have a mechanism for manually starting a task, for example for specifying a single component and its subcomponents. In such a case, the procedure list should only include procedures relevant for that task and ignore others.

2.4 Configuration Information Management

There are two orthogonal directions for changes in product modelling. The first one is the life cycle of a product. It consists, for example, of the following phases: Tendering, customer specification or order, configuration of the product, manufacturing, assembly, delivery, maintenance, modernization and finally demolishing. The second direction is the life of the product models.

The selling of the product starts with the product models available and active at that time. Then, basically, the product model, i.e., the laws the product will obey, is selected

and more or less fixed for the rest of the products life. Order specification, configuration, and so on, up to the demolishing, will be performed according to an instance of a product model at that time.

The product data related to that product has to be self-supporting. It has to carry all the needed information through all these stages. All the changes to the product should be reflected in that data, which suggest that the data should be in a general, standardized format. Would it become necessary to reconfigure the product, the configurator should be able to reconstruct the environment as it was when the product departed from the configurator. If the product models have evolved, the user has the choice of keeping the product independent of those changes or propagating the modification into the product. The latter means selecting a newer version of a product model to be the new model for the product. Usually the product must be changed to make it valid according to the newer configuration rules.

3 Configuration Data Model

3.1 Product Models and Configurations

We generalize the concept of a product model so that each configuration is based on some product model. Even a configuration which has been created "from scratch" is based on a product model with the general application domain rules as mentioned in Section 2.2. The product model provides a "skeleton", which is gradually refined during the configuration process. For example, configurations C1 and C2 in Figure 2 are based on the product model M5 (the meaning of the other lines in the figure will be explained shortly).

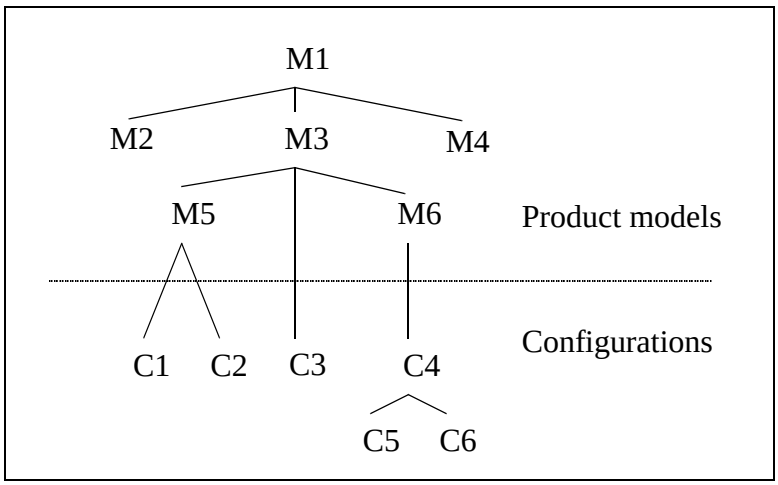

Figure 2: Product models and configurations

Configuration process begins with a customer specification. This specification is matched against the existing product models, and the most suitable one is selected. The

selected model provides constraints that restrict the set of possible component combinations, and procedures that help the designer to create good configurations.

The amount of interaction between the designer and the configurator system during the configuration process depends on the product model. Some product models contain procedures which create a configuration automatically from a specification. These models are called *standard product models*, and the corresponding configurations are *standard products*.

Typically a standard product model specifies most—if not all—components of the configuration. The model must only include rules for computing appropriate attribute values for the pre-selected components according to the specification. Often some attribute values have been pre-computed for the cases covered by the standard product model.

Many product models share common properties. The models are therefore organized in a *specialization* (or inheritance) hierarchy. In Figure 2, product model M1 is the root of the model hierarchy and includes the common properties of all lifts. Model M1 could, for example, include a constraint which says that all lifts must have a car as a component.

Model M3 is a specialization of M1, which means that M3 has all properties (attributes, constraints, procedures, etc.) of M1, and some additional properties of its own. When a product model is considered to correspond to a set of possible configurations (i.e., not only the configurations which actually exist at a particular moment), the configurations of model M3 are a subset of the configurations of model M1.

A configuration need not be based on a leaf node in the model hierarchy. For example, configuration C3 is based on model M3, which also used a basis for two more specialized models, M5 and M6.

Specialization can also arise among configurations. Continuing on Figure 2, configuration C4 is based on a non-standard product model M6. Two alternative designs of this configuration are being investigated. The designs are represented as configurations C5 and C6, both of which are specializations of C4. Note that from the point of view of C5 and C6, C4 can be regarded as another product model. It specifies certain properties, which are further refined in C5 and C6. Configuration C4 can, for example, introduce new attributes, whose values must be determined in C5 and C6.

These observations lead to one of the key ideas of our system: Product models and configurations are similar objects, which are manipulated with the same operations. All relationships between the objects in Figure 2 are similar specialization relationships.

Many object-oriented systems make a fundamental distinction between object classes (or types) and instances of the classes. In these terms product models could be regarded as classes, and configurations as their instances. However, this distinction seems unnecessary in our domain. In the same way that a particular product model is a specialization of a more general lift model, a particular lift configuration is a specialization of a product model.

3.2 Prototype Object Model

As the previous section shows, we represent product models and configurations with a prototype-based object model [9], [15]. Prototype model has some interesting properties, especially for engineering data modeling (see [2], [3], [5]). We do not know any study trying to elaborate the prototype model to support the creation and long term maintenance of complex product models discussed in this paper. Some properties of our object model are presented below, more details can be found in [14]. At this point we are convinced that the benefits of dropping the class-instance model are greater than the losses resulting from it.

3.2.1 Object Inheritance

All models and configurations in Figure 2 are objects. Each object is a collection of *object elements*, which comprise *attribute declarations*, *attribute assignments, constraints* for checking object validity, and *procedures* for creating valid objects.

Each object is a specialization of its *parent object*. The parent object must be specified when a new object is created; an important property of our model are tree transformations which change an object to have a new parent.

The relationships *child*, *ancestor* and *descendant* are defined in the normal way from the parent relationship. An object *inherits* all elements (declarations, assignments and constraints) of its ancestors. Moreover, an object is said to *possess* all elements that it either contains or inherits.

3.2.2 Attributes

An object can contain a number of *attribute declarations*. Each declaration specifies the name and type of an attribute.

If an object possesses the declaration of an attribute, an attribute assignment to the attribute can be added to the object. The assignment specifies the name of an attribute and a value, which must conform to the attribute type in the declaration.

The value of a particular attribute in a particular object is read from the first assignment to the attribute on the path of objects from the given object towards the root object along the parent links. The value assigned to an attribute in an object can thus be regarded as a default value for the attribute in the descendant objects because the assignment is inherited, but any descendant can add an assignment to the same attribute. For example, a product model can specify default attribute values, which are inherited by configurations.

3.2.3 Specialization Tree Transformations

The model gains much of its flexibility from the possibility of changing an object to have a new parent as long as one does not attempt to create a cycle in the object hierarchy. The

effects of a parent change depend on the relationship between the original and the new parent.

When the new parent of an object is a descendant of the old parent, the object is *specialized*. It inherits more attributes and constraints than before and represents a smaller set of possible objects.

In Figure 3, a lift configuration is represented with the object *order-123*. Originally the configuration is only specified to be some kind of a hydraulic lift. A general product model for hydraulic lifts is therefore selected as the parent for *order-123* when the object is created.

Later the designer selects one of the standard product models, HLX, as a basis for the configuration. This standard model includes an automatic ventilation system and accordingly object HLX contains a declaration for attribute *fan_power.* After *order-123* is changed to have HLX as the parent, an assignment to *fan_power* can be added to *order-123*.

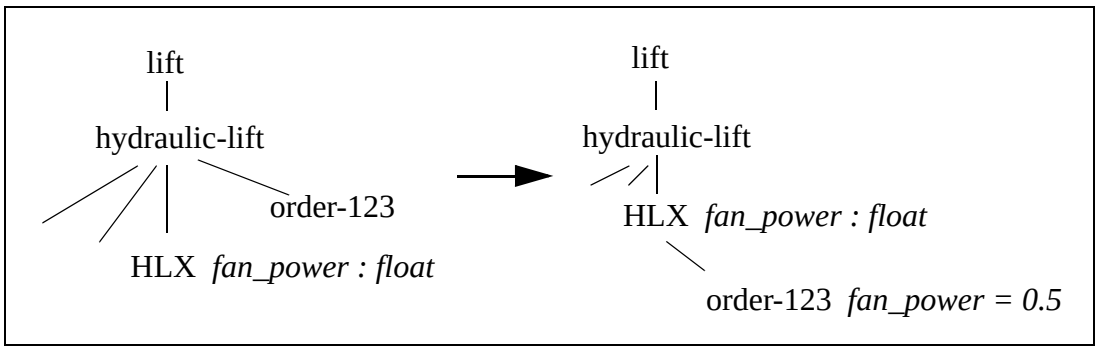

Figure 3: Object specialization

An object can also be changed to have a new parent which is an ancestor of the original parent. As a result of this *generalization*, the object will inherit fewer attributes and other elements than before.

To preserve some of the semantics of the object, we copy the intervening elements to the object. The generalization of an object does not change any attribute values or validity constraints. However, some attributes and constraints become "local properties" that can be modified without affecting other objects (except of course the descendants of the modified object).

Figure 4 shows the same objects as Figure 3. Lift has a new attribute *max_load*, which is constrained in the standard model HLX to have a value between 100 and 500. However, the maximum load of *order-123* has been assigned value 600. Since this violates the inherited constraint, the configuration no longer belongs to the set of valid lifts as specified by the standard model HLX.

The designer therefore generalizes the configuration into a hydraulic lift. The configuration is still invalid but the violated constraint is now local and can be relaxed or deleted in the configuration. The declaration of *fan_power* is automatically copied from HLX to *order-123*, allowing the configuration to preserve the value assigned to the attribute.

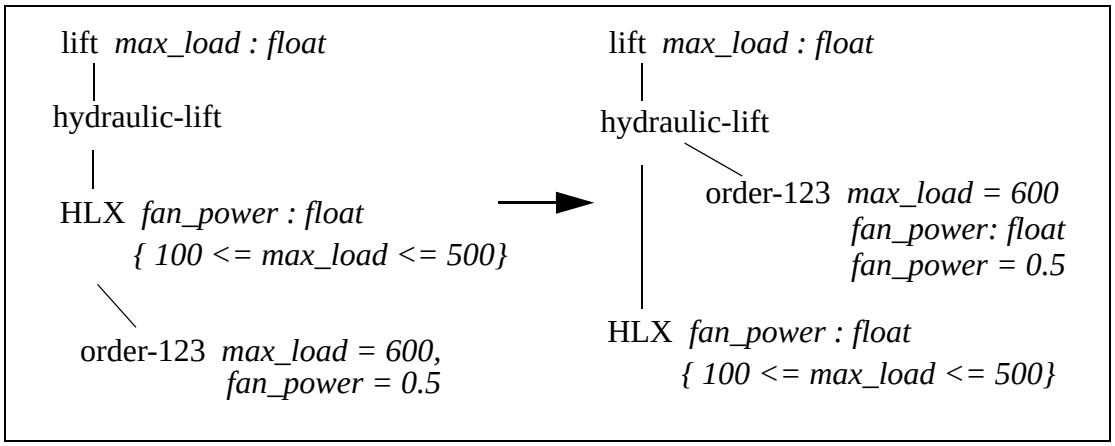

Figure 4: Object generalization

In this example, *lift* and *hydraulic-lift* are regarded as product models while *order-123* is a configuration. Nevertheless, this distinction only reflects the intended use of the objects. Any "configuration object" can be used as a parent for a new object, thus promoting the configuration to a "product model".

We have described the evolution of a configuration. Note that the same mechanism can be used for making new product models. The generalization step in Figure 4 could create a new product model on the basis of the existing model HLX.

We also model versions of an object as its specializations. The same mechanism can be used for versions of both configurations and product models. For example, two versions of the model HLX can be represented an its children. Anything these versions have in common is placed in HLX and the differences are modeled in the versions. Generic references to these versions are simply references to HLX. Similar ideas have also been proposed by Batory and Kim [2] and by Chang and Katz [3].

3.2.4 Components

The model defines two kinds of relationships between objects: An object can be the parent of another object as described above, or a an object can be a component of another object. The latter relationship is represented by means of attributes that have references to other objects as their values and behave in a special way during object specialization and generalization. There are also special constraints for describing the allowed component structure of valid configurations [14].

4 Future Work

In an earlier research project we developed a prototype Engineering Document Management System (EDMS), that functions as a repository for all kinds of engineering documents [13]. The configurator system should connect the final product descriptions to corresponding drawings, technical specifications etc. stored in the EDMS.

The final result of a configuration process is a detailed description of the product to be manufactured. The process where the product information is extracted from the configurator is called *instantiation*. This instantiation generates self-supporting product data which includes all constraints and procedures that were associated with the product in the configurator.

We make the instantiated product self-supporting in order to allow fully functional processing later. The instantiated product should have enough information so that the configuration process can be restarted at any time in our configurator. The product can also be taken to another system for reprocessing. In fact, we allow instantiation of any information in our configurator, so that also product models can be transferred outside the system.

We would like to emphasize that we start this modelling from a real world problem. We try to build a concise data model for this domain using the prototype approach as a basis. We are not too worried about the implementation details at this stage. After we have clarified the needs of KONE Elevators and developed a framework for modelling the needed aspects, we believe we can implement a system having the needed functionality. One possible solutions is to build it as a separate toolkit on top of an existing configuration generation system, e.g., the D++ of the Design Power, Inc. [6], [12].

We are also investigating the possibilities for representing the knowledge in a general format which allows any system supporting the transfer format to manipulate instantiated products. This approach would allow our system to serve as a maintenance tool for product models and knowledge. The configurations would be generated with existing expert systems using data retrieved from our system.

5 Conclusions

We have presented our research directions for solving a problem of configuration information management. Management of such information includes the management of various kinds of changes on both the data and the data model.

We identified the management of a distributed configuration process and long term management of configuration information as the main problems to be solved. These are inherently independent of each other.

The former problem involves a configuration process where multiple partners make decisions concerning the outcome of the process. They have to have a mechanism of reaching a consensus and retaining it after a change.

The latter problem is not well covered in existing configuration systems, and the failure to manage the configuration information when product models, components, design principles etc. change, makes many configurators useless when time passes. The effort to make the configurator functional again can be comparable to starting a new configuration management project.

6 Acknowledgments

This project is carried out in close co-operation with KONE Elevators, and we would like to thank Asko Martio, Juha Mäkäräinen, Simo Lassila, and Rauno Nousiainen for their time and advice. We would also like to thank Juha Tiihonen for his valuable comments on this paper. The project has been partly funded by the Finnish Technology Development Centre (TEKES). The project is a part of the SIMSON Technology Program of the Finnish Metal and Engineering Industry (FIMET).

7 References

[1] Virginia E. Barker and Dennis E. O'Connor. Expert systems for configuration at Digital: XCON and beyond. *Communications of the ACM*, 32(3):298–318, March 1989.

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

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

[4] Bill Curtis, Marc I. Kellner, and Jim Over. Process modeling. *Communications of the ACM*, 35(9):75–90, September 1992.

[5] Adrian Demaid and John Zucker. Prototype-oriented representation of engineering design knowledge. *Artificial Intelligence in Engineering*, pages 47–61, 7 1992.

[6] Design Power Inc., Cupertino, CA. *Design++ Reference manual*, 2.1 edition, 1992.

[7] Tsuneyoshi Katsuama, Hirokazu Taki, Hidekazu Tsuji, Akihito Naito, Motonori Yoshida, and Kihatirou Ohnishi. An expert system for elevator design. In *Proc. of the World Congress on Expert Systems 1991*, pages 36–45, 1991.

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

[9] Henry Lieberman. Using prototype objects to implement shared behavior in object oriented systems. In *Conference on Object-Oriented Programming Systems and Languages (OOPSLA)*, pages 214–223, 1986.

[10] Sandra Marcus, Jeffrey Stout, and John McDermott. VT: An expert elevator design that uses knowledge-based backtracking. In *Artificial Intelligence in Engineering Design. Volume I*, chapter 11, pages 317–355. Academic Press Inc., 1992.

[11] John McDermott. R1 ("XCON") at age 12: Lessons from an elementary school achiever. *Artificial Intelligence*, 59(1–2):241–247, February 1993.

[12] David J. Mishelevich, Matti Kataja-mäki, Tapio Karras, Alan Axworthy, Hannu Lehtimäki, Asko Riitahuhta, and Raymond E. Levitt. An open-architecture approach to knowledge-based CAD. In *Artificial Intelligence in Engineering Design. Volume III*, chapter 5, pages 125–178. Academic Press Inc., 1992.

[13] Hannu Peltonen. EDMS engineering data management system—architecture and concepts. Technical Report TKO-B59, Helsinki University of Technology, Laboratory of Information Processing Science, 1993.

[14] Hannu Peltonen, Tomi Männistö, Reijo Sulonen, and Kari Alho. An object model for evolutionary configuration management. Technical Report TKO-B93, Helsinki University of Technology, Laboratory of Information Processing Science, 1993.

[15] Lynn Andrea Stein. Delegation is inheritance. In *Conference on Object-Oriented Programming Systems and Languages (OOPSLA)*, pages 138–146, 1987.

[16] Gregg R. Yost. Configuring elevator systems. Technical report, Digital Equipment Corporation, 1992.