Towards Evolving Configuration Models

Thorsten Krebs[1], Lothar Hotz[2], Christoph Ranze[3], and Guido Vehring[3]

[1] LKI, Fachbereich Informatik, Universität Hamburg
Hamburg, Germany, 22527
krebs@informatik.uni-hamburg.de
[2] HITeC c/o Fachbereich Informatik, Universität Hamburg
Hamburg, Germany, 22527
hotz@informatik.uni-hamburg.de
[3] encoway GmbH & Co. KG
Bremen, Germany, 28359 {ranze|vehring}@encoway.de

Abstract. Configuration models describe commonality and variability as well as restrictions within and between components of a product domain. Whenever new versions or variants of products are brought to the market, changes to the configuration model (i.e. evolution) are inevitable. In this paper we discuss what kind of modifications are necessary for evolution and the potential effects on the model's consistency and on products during and after the configuration process. Validity intervals are presented as a promising approach to incorporate a versioning history for knowledge entities in configuration models.

1 Introduction

Knowledge-based configuration has been successfully applied to the configuration of technical systems [5] and is currently being applied to software configuration [9, 1]. There are different configuration approaches; e.g. rule-based, constraint-based and resource-based. In this paper we focus on evolution of structure-oriented configuration models. Two well known configurators using methods of structure-oriented configuration are KONWERK [4] and EngCon [12]. Structure-oriented configuration is based on configuration models representing entities of the "real world" through *concepts*. Such concepts contain a name and an arbitrary number of properties, i.e. *parameters* and *relations* to other concepts. With these relations, taxonomic and compositional hierarchies are modeled. Furthermore, *constraints* between concepts and their parameters can be defined[4].

Configuration models are generated before configuration can take place. A model describes the set of all configurable products. This is the descriptive part of product family *domain engineering* where an asset store is created. Products are derived using these assets during *application engineering* [2]. In a *knowledge-based* approach, the application engineering is supported by configuration models. This implies that only a description of the product and its components is generated, not the product instance itself. Thus, the configuration model contains descriptions for domain assets of the product family.

[4] For more details on the modeling facilities of structure-oriented configuration see also [3, 13]

Evolution is inevitable in the life-cycle of configurable products. Whenever new versions or variants of products are brought to the market, changes to the configuration model need to be synchronized with the ”real world”. Incorporating new versions and variants of products means extending the product family. The knowledge base describes all members of a product family that can be derived using knowledge-based configuration techniques. Thus, the configuration model describes admissible configurations and has to be kept up-to-date with the real products and product components.

It is desirable to be able to proceed with already existing (partial) configurations at any time in the future. Problems can occur when the configuration model was modified in the meanwhile. Explanations about affected changes in the model and their impacts (e.g. errors while loading the configuration) then can help the modeler. Furthermore, suggestions for repairing the outdated configuration are desirable – such a repair can be implemented interactively or automatic. When components of a former solution no longer exist, recommendations for compensation should be given.

The remainder of this paper is organized as follows. First, in Section 2 we give a survey on evolution. The influence of evolution on the configuration model and process is discussed in Section 3. The approach of validity intervals is introduced in Section 4. In Section 5, related work is presented.

2 Evolution

Throughout the lifetime of a product family, new requirements arise that require evolution. This can be driven by different factors like advancing technical abilities for realizing certain functionality or through evolving customer requirements that yield to changes in products or product components. Evolution is almost impossible to predict in the modeling phase. But it is common that future evolution is anticipated to a certain extend and therefore the product family design is prepared for this e.g. by modeling planned features [6]. But eventually there are unpredicted requirements (like bug fixes) or other situations where planning evolution is not practical. Evolution can be divided into the following categories [7, 10]:

During Domain Engineering For structure-oriented configuration, evolution during domain engineering is the task of extending the configuration model, i.e. modeling new variants and versions of components or modifying existing ones. This can be described as *preventative* evolution and is concerned with system improvements and correcting errors before problems in usage can be detected by the user.
Instead of narrowing the model, broader value ranges for parameters and relations can be modeled a priori. Thus, more configurations are covered by the model and less evolution tasks have to be performed. In a broader model it is not necessary to have all configurable assets implemented in the asset store. Moreover, information about the effort needed for realizing the final product can be computed from the model.

During Application Engineering During application engineering, usually the model is fixed for structure-oriented configuration techniques. This means, possibilities for dynamically modifying the configuration model have to be taken into account.

This kind of evolution is called *adaptive* or *perfective*. Adaptive modifications include addition of functionality, changing functionality and support for new platforms. Perfective evolution is concerned with advancing functionality and system / performance improvements.

Instead of configuring given concepts, evolution of the model could be included in the configuration process (started by the user or indicated by the system). This means, solutions beyond descriptions provided by the model are possible. During the modeling phase (which is part of domain engineering), places where evolution is possible can already be defined. E.g. if two incompatible concepts are chosen, a conflict resolution might be to develop and integrate a new concept.

During Maintenance Evolution can also occur after time-to-market. Using a product (i.e. during maintenance) e.g. a bug can be found. This has to be corrected in a new product version (bug-fix). Different scenarios like patching (only possible for software), re-design and new development of the product or for just one component of this product are possible solutions.

Corrective evolution cannot be anticipated and thus has to be done after bringing the product to the market.

In all of the above scenarios, for structure-oriented configuration the main focus lies on extending the configuration model. Knowledge acquisition is the central aspect for new concept descriptions as well as modifying existing concepts during domain engineering, application engineering and maintenance. So far, knowledge acquisition has only been taken into account for building a fixed configuration model to use for product derivation. The same acquisition techniques can also be used for identifying and modeling new and changing existing concept definitions. In this task, a special focus has to be laid on consistency because changes to the model can have impacts on the currently developed partial configuration. This will be further discussed in the following sections.

3 Evolution within the Configuration Process

After a survey about evolution in general, in this section we transfer these ideas to the process of knowledge-based configuration. E.g. when at a point in the configuration process a conflict is detected (i.e. the task specification does not go together with the configuration model), usually the task is adapted to the model by conflict resolution methods like backtracking and providing different input for the inference machine. Another way to solve this is to state that the task specification is correct and to modify the configuration model. This decision however has to be seen critical. Companies use configuration models for ensuring consistency and buildability of the product solution. This can no longer be guaranteed when the model is modified during the configuration process.

The aspect of evolution within the configuration process can be seen as a kind of *innovative configuration*[5]. We see innovative configuration not as an absolute term but as a relative one - relative to a configuration model. Innovation related to a model is given

[5] A survey on innovative configuration is given in [8, 3]

if the solution to a task specification is not covered by this model - i.e. it lies outside of the solutions this configuration model describes. To supply a configuration process where evolution of the domain model is a basic task, we expect to apply methods known in innovative configuration to be used [7].

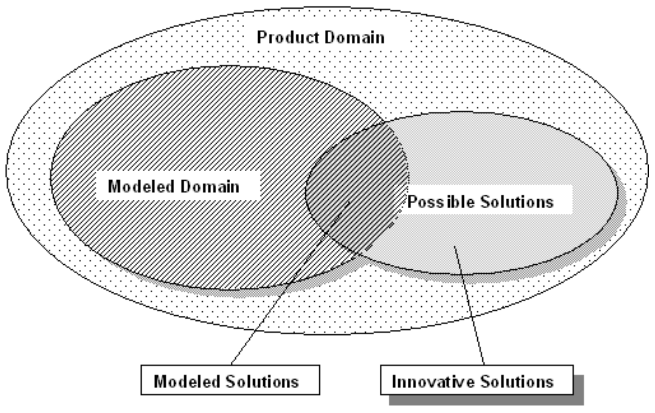

Fig. 1. Innovative Configuration

As depicted in Figure 1, a knowledge model only covers part of the *product domain*. The *possible solution space* is that part of the product domain that conforms to the task specification. The intersection of the *modeled domain* and the possible solution space is the *modeled solution space* – i.e. all admissible configurations that are solvable with routine configuration. Other solutions that lie outside of the modeled domain are also admissible solutions to the task specification, but are only reachable with innovative configuration techniques. These are called *innovative solutions*.

Reality always includes solutions that are not covered by the used configuration model, because a part of the solution space can be unknown or not modeled. Thus, in routine configuration, enhancements and modifications of the domain have to be concerned isolated from concrete product solutions in the phase of knowledge acquisition. Further, continuous enhancements in the product domain make it hard to maintain configuration models. Because modeling configuration knowledge can be seen as a specific kind of evolution, by supporting dynamic knowledge modeling, (at least a part of) evolution is already covered.

3.1 Evolution of Configuration Models

Evolution of the configuration model can have different effects on product configurations and consistency of the model. Consistency is defined as follows:

Specialization-related: given a super- and a subconcept, all values of the subconcept's properties have to be subsets of the corresponding property (identified by name) of the superconcept.

Composition-related: given an aggregate and its parts, each part has to be defined as a concept.

Constraint-related: given a constraint and the participating concept properties, the constraint may only use value ranges defined in the model. Furthermore, only subsets of values of the concept properties are allowed as propagation results.

There are two basic functions for evolving knowledge models: *add* and *delete*. Other modifications, e.g. changing the value range of a parameter can be split into deleting the old and adding the new value. The following list applies these basic functions to given knowledge entities.[6]

Addition of new parameters Adding new parameters is unproblematic with regard to consistency of the configuration model. But inheritance needs to be addressed.

Deletion of existing parameters Deleting a parameter may in some cases affect the consistency of the configuration model. If this parameter has relations to other parameters or concepts - i.e. if it is participating in a constraint, consistency of the model can no longer be guaranteed. Moreover, if the model stays consistent, the new value may still lead to a different configuration solution by selection of different components through varying constraint computation. But in any case inheritance has to be taken into account like in the previous aspect.

Addition of new specializations Adding a new specialization means establishing a new taxonomic relation between two concepts. Inheritance of the corresponding parameters and relations has to be considered. Concept instances of a configuration solution may be of a different type after adding new specializations.

Deletion of existing specializations Deleting a specialization means removing an existing taxonomic relation between two concepts. Deleting a specialization, the former subconcept no longer has a superconcept, which is not valid since the configuration model is described in a tree, not in a forest. There are two possibilities to solve this situation:

1. The former subconcept has to be moved under another superconcept. This means, adding a new specialization is performed.
2. The former subconcept is deleted. This can be split into deleting its parameters and relations accordingly.

After deletion of a specialization, the solution to a configuration task can be different since needed concepts may no longer exist.

Addition of new decompositions Adding a new decomposition means establishing a new compositional relation between an arbitrary number of concepts. New has-parts and part-of relations have to be integrated into the superconcept and the subconcepts respectively. Configuration solutions may be different; they may contain more concepts than before.

Deletion of existing decompositions Deleting a decomposition means removing an existing compositional relation. Multiple parts can be involved in this task. Deleting decompositions always leads to necessary modifications in both, the has-parts relation of the aggregate and the part-of relation of the parts. If the part is no longer

[6] All points only hold under the assumption that additions to the configuration model are correct - i.e. consistent with the rest of the model.

part-of an aggregate, the relation has to deleted accordingly. If an aggregate has no other parts belonging to that has-parts relation, this relation has to be deleted. In other cases, the cardinality of the concerned parts has to be aligned. A configuration solution may contain less concepts than before.

Addition of new constraints Adding a constraint does not have any effects on the consistency of the configuration model when participating properties exist in the model and calculated values are inside of valid ranges for these properties. Constraints describe restrictions between concepts and therefore can affect configuration decisions. Thus, the solution to a task specification can contain different concepts and values for their parameters than before.

Deletion of existing constraints Deleting an existing constraint also does not cause any harm to the consistency of the configuration model. But just like adding constraints, also in this case the solution for the same task specification may be different because other configuration decisions are possible.

Examples for combined operations on parameters, relations, concepts and constraints are listed in the following:

– Modification of a parameter value can be split into deleting the old and adding the new value.
– Modification of relations can have very different semantics. The part-of relation and the has-parts relation are strongly connected. Modifying the aggregate of a given concept (i.e. changing the name in a part-of relation) e.g. would also require modifying the relevant has-parts relation; remove an entry from the old and add an entry to the new superconcept. But all such modifications to relations can be split into removing and adding single relations.
– Addition of a concept can be split into adding a specialization and possibly adding new parameters and relations. The same goes for deletion of a concept.
– Modification of a concept can be split into deleting and adding a concept. Analogous, modification of a constraint can be split into deletion and addition.

Changes to the procedural knowledge model do not necessarily effect the configuration solutions. The order of configuration steps is transitive - i.e when the same input is given for all configuration decisions, the order can not change values of the outcome.

Transitivity of the configuration process only holds, if user decisions and system-sided inferences (e.g. taxonomic inferences or constraint propagation) provide the same input for all configuration decisions. This is rarely the case; e.g. when automatic mechanisms like stored command protocols from previous configurations are used as input. Thus, the order of configuration decisions can lead to different solutions when the user is confronted with different value ranges at the same decision or with decisions he otherwise would not be confronted with at all.

3.2 Degrees of Modifications

In the previous section we addressed possible modifications for evolution of configuration models and how complex tasks can be split into the single functions add and delete.

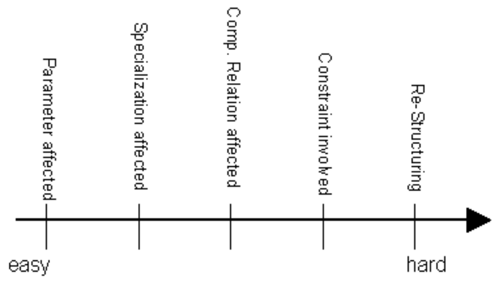

Fig. 2. Degrees of Modifications

In this section we discuss which effects changes can have on the configuration model and on solutions and how these can be handled for further usability of the model.

We differentiate between degrees of modifications that range from easy handling of the situation up to effects that make configuration models hardly maintainable. In Figure 2 we show which kind of modifications to the configuration model affect its consistency and maintainability in different gradations. Their effects on the configuration model and ideas how to handle these situations are presented in the following:

Easy to Handle An easy to handle evolution situation is given when e.g. a concept that has no further subconcepts or a parameter of such a concept is added, modified or deleted. As long as no relations (taxonomic and compositional) or constraints are involved in the evolution task, the modifications do not cause inconsistencies or entail the necessity of further modifications.

Unproblematic Changes of concepts and their parameters can have an impact on subconcepts of the affected concept. Through inheritance, subconcepts also own the parameters and relations of their superconcepts. Thus, modifications have to be processed through the taxonomic hierarchy down to concepts with no further subconcepts. In general, compositional relations can concern more than one *branch* of the knowledge tree (i.e. different kinds of subconcepts) and therefore all branches with concepts participating in the affected compositional relation have to be processed.

Inconsistencies may arise e.g. when a property value is being narrowed down in the sense that the value of the same parameter in a subconcept is no longer a subset of the superconcept's parameter value. In this case two solutions are possible to resolve the inconsistency:

1. The property value of the subconcept has to be modified corresponding to the property value of the superconcept.
2. The property value of the superconcept always has to be the union of the values in all subconcepts and therefore has to be repaired accordingly.

Problematic More problematic evolution situations arise when constraints are involved in modifying parameters or relations or when the constraints are indicted themselves. In either case, consistency of the model can no longer be guaranteed. This is a result of more complex dependencies within constraints than within taxonomic or compositional relations. Constraints are not bound to one branch of the knowledge tree and more than one constraint can be applied to a concept's property. This makes it hard for the human modeler to track effects of evolutionary modifications and gives raise to automated methods for insuring model consistency.

At this point, two situations for repairing the configuration model have to be treated distinctly:

1. The value of a property participating in a constraint has been modified or such a property has been deleted. In this case, the affected constraint has to be repaired. For the deletion of a parameter this can mean that a constraint is no longer needed or that a different parameter can be inserted instead. For modified parameter values, the impact on constraint computation and on other participating values has to be addressed. This includes changes within the concerned constraint and value changes of other properties.
2. A constraint has been added or modified. Impacts of the constraint on corresponding concept properties have to be taken into account. For inconsistent situations either the constraint or the concerned property values have to be modified in order to regain consistency of the configuration model.

Critical Evolution tasks can influence the configuration model so deeply that it is hardly possible to repair it by a number of simple actions. This is the case when a model is completely restructured, e.g. when coherent concepts are moved - i.e. a subtree is placed under a different superconcept. Guaranteeing consistency of the configuration model is hard to achieve for this situation. Restructuring the concepts implies changing the design of configurable products. Hence, no former consistency assumption can hold and all configurations have to start from scratch. Moreover, three critical effects of restructuring a configuration model can be identified:

1. Inheritance is hard to maintain for permuting complete subtrees. Deleting properties that are no longer valid because they are no longer inherited by the former superconcept and adding all properties now inherited by the new superconcept have to be done separately for every movement. Otherwise, afterwards the needed information is lost.
2. Constraints can be affected by this evolution task. Because restructuring the concepts means formalizing a new product platform context, affected constraints cannot simply switch the participating concepts due to the fact that the new context of the model may not be covered by this constraint - i.e. the evolution task invalidates this constraint.
3. Configuration solutions are not only slightly changed but represent a substantially different product. Comparisons to former configurations (e.g. reference configurations) are no longer possible. Also, products that are already on the market cannot be compared to the current configuration model - e.g. for detecting spots where upgrades can be performed etc.

Changes in the product architecture are wanted - they are the goal of evolution tasks.

Therefore it is not a problem that starting a configuration with the same task specification, a different solution is generated with an evolved model. But this has the effect that reference configurations are not necessarily still valid. Stored former configuration solutions are sometimes utilized for checking if the configuration model stays consistent after applying evolution tasks. For the case that the product domain has deeply changed and reference configurations can no longer be used, they have to be repaired or new ones have to be created.

A possibility to go round this task is applying versioning concepts to knowledge entities in configuration models. This is further elaborated in the following section.

4 Validity Intervals

The configuration model can be stored in different versions. Multiple modifications together form the transfer into a new major version. This can be realized very easily by defining versions from time to time. When a configuration ”knows” the version of the model it was created with, it can be loaded at any time in the future. But this mechanism is not very flexible and cannot give hints about changes of the model and why the configuration is inconsistent with a newer version of the model. Using *validity intervals*, every modification can be traced individually. This makes it possible to explain why inconsistencies appear and maybe even how they can be repaired. Another advantage is the ability to perform changes before they are valid. E.g. when new safety regulations or ordinances are given for a date in the future, the model can be prepared beforehand.

Knowledge entities are annotated with a time interval in which they are valid. The lower bound of a validity interval represents the point in time when the knowledge entity was added to the model (i.e. the start of validity) and the upper bound represents the point in time the entity was deleted (i.e. the end of validity).

```
 Concept
   name: Motor
   superconcept: Device
   parameters:
Power [5 25] <-inf .. 2003-03-18@14:33:26>
Power [10 30] <2003-03-18@14:33:27 .. inf>
```

Fig. 3. Definition of a Modified Parameter

In Figure 3, the usage of validity intervals is introduced. The key words `-inf` and `inf` represent unknown (i.e. infinite) time restrictions for the lower and the upper bound respectively. All knowledge entities are implicitly assigned with the interval `<-inf .. inf>` when nothing else is modeled. Therefore, existing knowledge does not have to be converted for usage of validity intervals. The fidelity of the interval bounds can be chosen to best fit particular domains. While sometimes the day is exact enough, for other domains the minute or even the second of modifications might be of interest.

This approach is in line with the splitting of combined evolution operations to the basic functions *add* and *delete* (see Section 3.1). Modification of a parameter value e.g. entails that two parameters with the same name and non-overlapping validity intervals exist within the same concept definition.

Two aspects are of particular interest concerning usage of validity intervals within the configuration process:

1. Loading an existing configuration is possible because all required concepts are given. In case of a conflict, it is feasible to determine whether a change of the model is responsible. Therefore the date of the configuration has to be compared to the validity entries. Further techniques can yield to repairing the configuration.
2. It has to be addressed if it should be possible to use expired configuration knowledge. This may be reasonable e.g. when a product is no longer fabricated but there still is a remainder in storage.

One major advantage of this approach is the introduction of *configuration model compilation*. This means, a configuration model containing only valid knowledge entities can be generated before configuration starts. Moreover, this can be done for any given time stamp, e.g. the date of a reference configuration.

Another aspect that has to be considered is monotonous growth of the configuration model. Expendable entries can be permanently deleted by user decision, automated expiration by date or automated deletion by memory usage. In the latter two cases, a notification to the modeler is desirable. Deletion can be initiated by event (e.g at the end of a modeling session) or by date (e.g. one a week or month).

5 Related Work

[11] also address evolution of configuration models. The characteristics that distinguish configurable products from traditional data modeling and management are addressed by concerning evolution of the schema and the instances. Existing data modeling approaches are stated to be inadequate. Therefore, *generic objects* are introduced as a collection of *versions*. *Effectivity* describes the time a version was or is representative for a generic object. In addition to specializations and compositions, in our approach we also have taken constraints (as a form of multilateral relation) into account.

6 Conclusion

In this paper we have focused on possible modifications to configuration models and their impacts on product configurations as well as the consistency of the model. Summarizing, modifying a parameter e.g. has few impacts on the model; only one concept is affected as long as this parameter value does not participate in constraint relations. Changing a specialization considers two concepts: the super- and the subconcept. Modifications to compositional relations entail changes in the aggregate and in all parts, which can distribute over an arbitrary number of knowledge subtrees. Modifying constraints can affect parameters and relations, thus comprises the impacts described above and re-structuring is seen as a compound of changing specializations and through inheritance also parameters, relations and constraints.

Validity intervals attached to arbitrary knowledge entities have been introduced to avoid the problem of having products (under development and already brought to the market), for which no consistent configuration model exists. Further benefits over simple versioning mechanisms have been pointed out.

References

1. T. Asikainen, T. Soininen, and T. Männistö, ‘Towards Managing Variability using Software Product Family Architecture Models and Product Configurators’, in *Proc. of Software Variability Management Workshop*, pp. 84–93, Groningen, The Netherlands, (February 13-14 2003).
2. J. Bosch, G. Florijn, D. Greefhorst, J. Kuusela, H. Obbink, and K. Pohl, ‘Variability Issues in Software Product Lines’, in *Proc. of the Fourth International Workshop on Product Family Engineering(PFE-4)*, Bilbao, Spain, (October 3-5 2001).
3. A. Günter, *Wissensbasiertes Konfigurieren*, Infix, St. Augustin, 1995.
4. A. Günter and L. Hotz, ‘KONWERK - A Domain Independent Configuration Tool’, *Configuration Papers from the AAAI Workshop*, 10–19, (July 19 1999).
5. A. Günter and C. Kühn, ‘Knowledge-based Configuration - Survey and Future Directions’, in *XPS-99: Knowledge Based Systems, Proceedings 5th Biannual German Conference on Knowledge Based Systems*, ed., F. Puppe, Springer Lecture Notes in Artificial Intelligence 1570, Würzburg, (March 3-5 1999).
6. A. Hein, J. MacGregor, and S. Thiel, ‘Configuring Software Product Line Features’, in *Proc. of ECOOP 2001 - Workshop on Feature Interaction in Composed systems*, Budapest, Hungary, (June, 18 2001).
7. L. Hotz, A. Günter, and T. Krebs, ‘A Knowledge-based Product Derivation Process and some Ideas how to Integrate Product Development (position paper)’, in *Proc. of Software Variability Management Workshop*, pp. 136–140, Groningen, The Netherlands, (February 13-14 2003).
8. L. Hotz and T. Vietze, ‘Innovatives Konfigurieren in technischen Domänen’, in *Proceedings: S. Biundo und W. Tank (Hrsg.): PuK-95 - Beiträge zum 9. Workshop Planen und Konfigurieren*, Kaiserslautern, Germany, (February 28 - March 1 1995). DFKI Saarbrücken.
9. T. Krebs, L. Hotz, and A. Günter, ‘Knowledge-based Configuration for Configuring Combined Hardware/Software Systems’, in *Proc. of 16. Workshop, Planen, Scheduling und Konfigurieren, Entwerfen (PuK2002)*, ed., J. Sauer, Freiburg, Germany, (October, 10-11 2002).
10. N. Loughran and R. Awais, ‘Supporting Evolution in Software using Frame Technology and Aspect Orientation’, in *Proc. of Software Variability Management Workshop*, pp. 126–129, Groningen, The Netherlands, (February 13-14 2003).
11. T. Männistö and R. Sulonen, ‘Evolution of Schema and Individuals of Configurable Products’, in *Proc. of ECDM’99 - Workshop on Evolution and Change in Data Management*, Versailles, France, (November 15-18 1999). Springer Verlag.
12. K.C. Ranze, T. Scholz, T. Wagner, A. Günter, O. Herzog, O. Hollmann, C. Schlieder, and V. Arlt, ‘A Structure-based Configuration Tool: Drive Solution Designer DSD’, *14. Conf. Innovative Applications of AI*, (2002).

13. T. Soininen, J. Tiihonen, T. Männistö, and R. Sulonen, ‘Towards a General Ontology of Configuration’, *Artificial Intelligence for Engineering Design, Analysis and Manufacturing (1998)*, *12*, 357–372, (1998).