# Principles of structuring complex configuration models using UML as domain oriented modeling language

**Alexander Felfernig, Dietmar Jannach, Markus Zanker**
Universität Klagenfurt
Universitätsstrasse 65
A-9020 Klagenfurt, Austria
+43 463 2700 6208
{alf, dietmar, markus}@ifit.uni-klu.ac.at

**ABSTRACT**

Shorter product cycles, lower prices and customer individual production are the main reasons for the preceding success of product configuration systems in various application domains (e.g. telecommunication industry, automotive industry, computer industry, or financial services). Configuration problems are a thriving application area for declarative knowledge representation that experiences a constant increase in size and complexity of knowledge bases. Since the resulting knowledge bases become increasingly large and complex, knowledge acquisition and maintenance are critical tasks and software development has to meet the challenges of developing highly adaptable software very rapidly. In this paper we show how to employ a standard object-oriented design language (UML – Unified Modeling Language) in order to design and structure complex configuration knowledge bases. In order to structure the configuration knowledge we employ standard structuring mechanisms provided by UML (packages, views) as well as context-specific models. Since domain experts mostly think in terms of concepts this approach leads to a more intuitive way of modeling configuration knowledge.

**Keywords**

Domain-oriented modeling environments, knowledge representation, structuring complex knowledge bases

## 1 INTRODUCTION

Mass customization [10], i.e. the customer individual production under pricing conditions of mass production created big challenges for the product development process. In this context knowledge-based product configuration systems (or configurators) support the technical expert or the sales representative to find a configuration of the product which conforms with technical or marketing restrictions and fulfills all the customer requirements. However, the increased use of knowledge-based configurators in various application domains as well as the increasingly complex tasks tackled by such systems ultimately lead to both the knowledge bases and the resulting configurations becoming larger and more complex (i.e., more components and constraints representing more involved component behavior). Due to shorter product cycles, lower prices of products, and higher customer demands and the fact, that configuration systems have to be developed concurrently with the configurable product, the construction and maintenance of the configuration knowledge bases have become a critical task.

In the case of product configuration, the available knowledge is distributed between many different people and organizational units (e.g., technical restrictions and marketing constraints) and changes very rapidly. Additionally, existing configuration systems often employ some high level (declarative) language and special terminology to represent the knowledge, which can not be formulated directly by the technical expert. Therefor, a knowledge engineer has to elicit and translate the knowledge into the representation of the actual tool.

In [2] a general high level modeling language from the area of Software Engineering (Unified Modeling Language - UML) [11] is employed for modeling product configuration knowledge bases. This approach is based on a commonly accepted terminology for configuration problems and can be used for automated compilation of knowledge bases, since the semantics of the individual modeling concepts are stated precisely. In addition, the graphical representation allows the model to be built on a conceptual level. The Unified Modeling Language is widely accepted in industrial software development processes and the resulting conceptual models alleviate the communication process between the people involved in the development process. The approach ([2]) uses the extensibility features of UML (stereotypes) to express the domain specific concepts for the product configuration domain, but does not use a proprietary modeling language.

The rest of the paper is organized as follows. In Section 2, we shortly review the technique to model configuration knowledge bases with UML (see [2]). Section 3 shows how the built-in structuring features of the UML can be employed. Section 4 discusses additional structuring

mechanisms beyond the scope of UML which increase the maintainability and understandability of the conceptual model. In Section 5 we show how to proceed in order in to build complex configuration models. Section 6 contains conclusions and related work.

## 2 CONSTRUCTION OF CONFIGURATION KNOWLEDGE BASES USING UML

Conventional bill-of-material representation mechanisms do not fulfill the requirements for modeling highly variant product structures. The set of possible product variants is described through different kinds of constraints stemming from user requirements, technical and economical requirements. Relations and restrictions between different parts can not be modeled with a bill-of-material representation. Existing knowledge-based configuration systems employ proprietary modeling concepts for representing the configuration knowledge, what causes a low degree of understandability and reusability. This challenge can be met by applying a wide spread design language for the representation of configuration knowledge.

Figure 1 shows how UML can be employed for modeling configurable products using the built-in extension mechanisms (stereotypes). The product structure is modeled through aggregation and generalization between the component types that form the final product. A configuration model created with the defined concepts can automatically be translated into an executable logical representation (e.g. a constraint satisfaction problem) (see [2] for further information). The following modeling concepts are typical for the product configuration domain [9]:

- Component types: These are the parts of which the final product is built of. Component types are characterized though attributes.
- Generalization: Component types with a similar structure are arranged in a generalization hierarchy and represent choices for the configurable product.
- Aggregation: These part-of structures are similar to classical bill-of-material structures. Multiplicities specify the number of subparts which the aggregate could consist of. In addition, we differentiate between *shared* and *compositional* aggregations.
- Resources: Sometimes a configuration problem can be seen as a resource balancing task, where some of the component types *produce* some resources and others are *consumers*. In the configured product these resources must be balanced.
- Connections and Ports: Additionally to the components which are part of the final configuration, sometimes information about the connections between the components is needed. Constraints on those connections can be expressed through *connected with* stereotypes which are represented through relations between component ports.
- Compatibility relations: Some types of components cannot be used together in a final configuration (they are *incompatible*). In other cases, the existence of one component type requires the existence of another special type in the configuration (*requires*).
- Additional constraints: Constraints on the product model, which can not be expressed graphically may be formulated using the language OCL (Object Constraint Language), which is an integral part of UML.

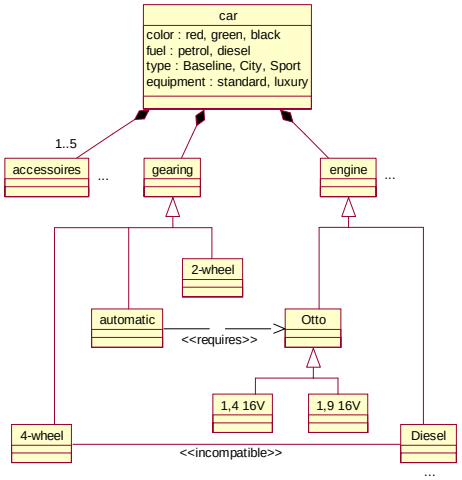


**Figure 1: product model of a car**

In [2] a logical model of a configuration problem is presented in terms of a logic theory. Additionally, translation rules are defined in order to translate the UML configuration model into an executable logical representation. The knowledge base is therefor represented and can be maintained on a conceptual level and can then be compiled into a executable representation. The resulting knowledge base is exploited by a general constraint solving engine.

The following paragraph contains an example for the logical representation of the part-of relationship between car and engine shown in Figure 1. Explanations concerning the translation rules for the other modeling concepts from the UML-model into the logical representation can be found in [2]. The logical representation is based on the component-port model described in [5].

```
/* available component-classes */
types={car, gearing, engine, 4-wheel, 2-wheel, automatic, Diesel, Otto, 1,4
16V, 1,9 16V, accessoires}.

/* component attributes */
attributes(car)={color, fuel, type, equipment}.
```

/* domains of attributes */
dom(car, color)={red, green, black}.
...

/* connection points of component classes */
ports(car)={gearing, engine, accessoires}.
ports(engine)={car}.
...

/* properties of the part-of relationship between car and engine */
type (ID, car) $\Leftrightarrow$ ($\exists$ ID1, Port) type(ID1, engine) $\wedge$ conn(ID, Port, ID1, car) $\wedge$ Port $\in$ {engine}.

type (ID, engine) $\Leftrightarrow$ $\exists$ (ID1, Port) type(ID1, car) $\wedge$ conn(ID, car, ID1, Port) $\wedge$ Port $\in$ {car}.

In addition to the constraints derived from the UML model as shown in Figure 1 further constraints are derived, e.g. 'if component X is connected to component Y then component Y is connected to component X too'. 'Components have a unique type' is another example for an application independent constraint. For further information see [4].

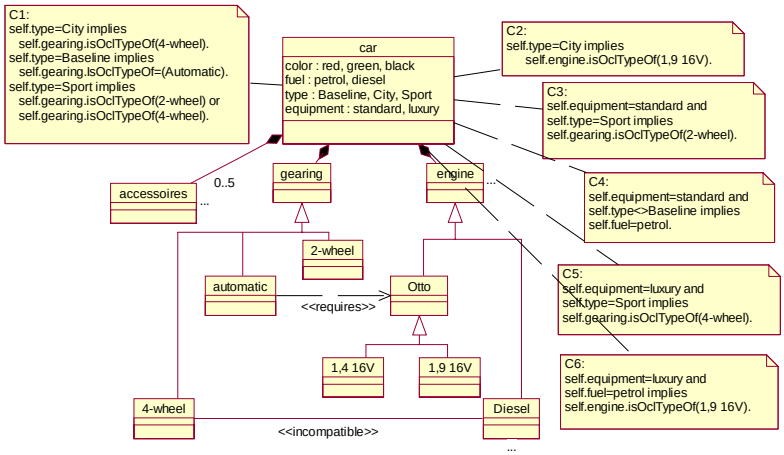


**Figure 2 : product model of Figure 1 with additional sales constraints.**

Employing the graphical notation discussed above results in quite understandable and maintainable models.

However, in cases when the product models become larger and more complex (due to many constraints and a large number of variants) even the graphical depiction of the problem becomes harder to maintain and understand. The configuration model of Figure 1 becomes quite unclear even if only simple sales constraints are added (see Figure 2). As a consequence of that structuring mechanisms are required which improve understandability and maintainability of the configuration model. In this paper we show, how the knowledge can be structured to cope with the increased knowledge base complexity using both the mechanisms predefined in UML and an approach using different views and contexts within the product model.

## 3 CONVENTIONAL STRUCTURING: EMPLOYMENT OF PACKAGES & VIEWS

The most straightforward approach to structure the knowledge base is the usage of the built-in structuring mechanisms of UML. A *package* is simply defined in UML as "a grouping of model elements. Packages themselves my be nested within other packages."

Figure 3 shows a simple engine package. Let us assume that the constraints which have to hold for the engine package have little impact on the other parts of the configuration model. It is therefor straightforward to model the structure of the engine in a package, as is it eventually may even be designed by a different person.

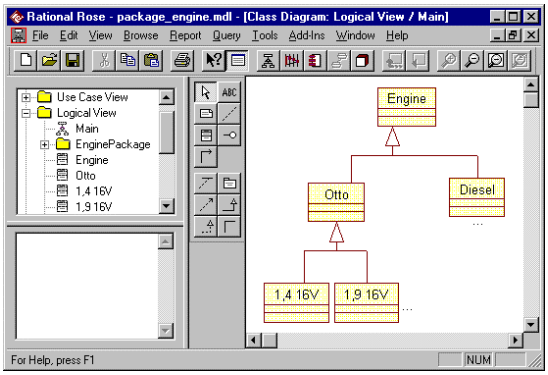


**Figure 3: model structuring through packages**

A package can be used to structure the solving process of the configurator as well. In some cases, different packages or parts of a configuration problem can be solved more or less independently. This information can be used to guide the configuration task which leads to smaller problem sizes and less computational complexity in the solving phase [4]. In addition, some form of order for the solving process or

an interactive configuration process can be expressed, i.e., we can define a configuration ordering between packages.

Most modeling tools provide different views which can be used to structure the knowledge. These views have no new semantics since the underlying model (classes, associations, etc.) is not affected when arranging the classes in different diagrams. For the domain of product configuration it is intuitive to define views which cover different aspects in a way that the resulting graphical depictions are not overloaded with information. In fact, a connection oriented view containing connections and ports as well as a view on the resources, which are mostly technical constraints can be hidden from a sales engineer who does not care about those constraints in the first place when he is defining marketing constraints.

Packages and views primarily support the grouping of components and relations into understandable chunks of knowledge. However, using these concepts complex constraints can not be reorganized into a more understandable representation. In Section 4 we present a context-based approach which enables an alternative representation of constraints while not changing the model semantics.

## 4 ROOT MODEL AND CONTEXT MODELS

The *root model* is the starting point for the specification of concept models. It includes all available component types the domain expert can use for constructing a configuration model. Additionally, only those constraints are included which are relevant independently of any context, i.e. it contains considerable less textual (OCL-) constraints as the former model shown in Figure 2. A context model is derived from the root model by selecting relevant components for the context. Those components of the root model not relevant for actual context are deleted, but naturally can be used for other context models (see Figure 5). Through restriction of attribute values, cardinalities, and generalization hierarchies, context-specific models are derived which could easier be manipulated by the domain expert, since the context specific constraints are graphically represented. The steps described can recursively be applied to the generated context specific model for generating further context specific models.

The root model consists of configuration relevant components including constraints which concern all product-families (here: C6). The root model is represented in Figure 4.

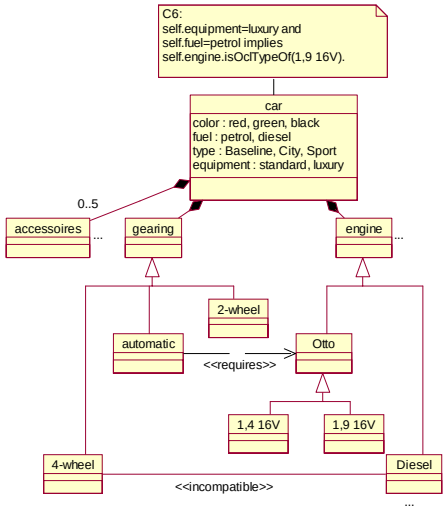


**Figure 4: root model**

For further discussions we use the example given in Figure 2. Constraints C1..C6 are constraints, which can hardly be maintained. Constraint C1 references the attribute ‘type’ with extensions Sport, Baseline, and City. Since these variants have several different constraints, it makes sense to do a separate modeling of these product families. The root model is the basis for constructing context specific models. Regarding the root model of Figure 4 we create different contexts depending on the car type (Baseline, City, or Sport).

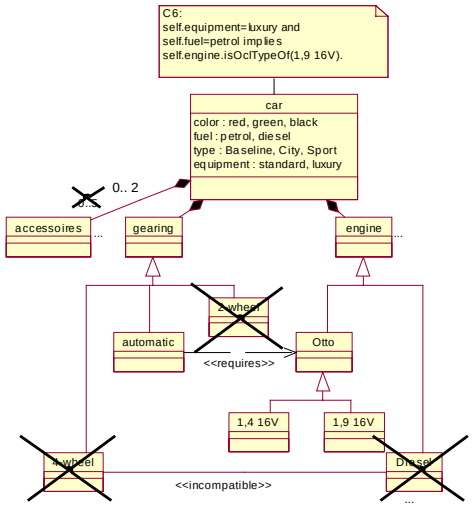


**Figure 5: building the Baseline context**

In order to create the Baseline context we select those components relevant for the context implicitly, i.e. we exclude components not relevant for this context. Further

we reduce the multiplicity of the part-of relation between accessoires and car. This restricted model represents the context Baseline where further restrictions can be formulated by the domain expert. Figure 6 shows the derived context model.

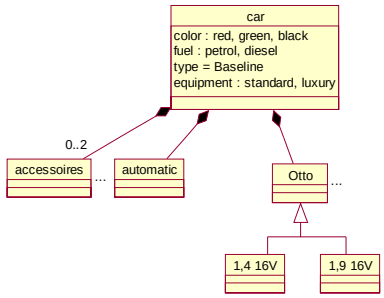


**Figure 6: car in context model „Baseline“.**

Figure 6 contains the context specific configuration model for the Baseline family, i.e. all relevant properties of a Baseline car are represented in this model. If we compare this model with the one in Figure 2 we detect a reduced set of constraints obtained by modeling the configuration knowledge for a specific product family. The textual representation of the constraints C1, C2, C3, and C5 is eliminated. Constraint C4 describes properties of Sport and City cars, but doesn't deal with properties of Baseline cars.

Because of that this constraint does not have to be considered. Constraint C6 is valid for all car types, because of that it is inherited to all context-dependent models. Figure 7 and Figure 8 contain context dependent models for Sport and City cars. Comparing these models with the configuration model in Figure 2 we detect a reduction of the textually represented constraints.

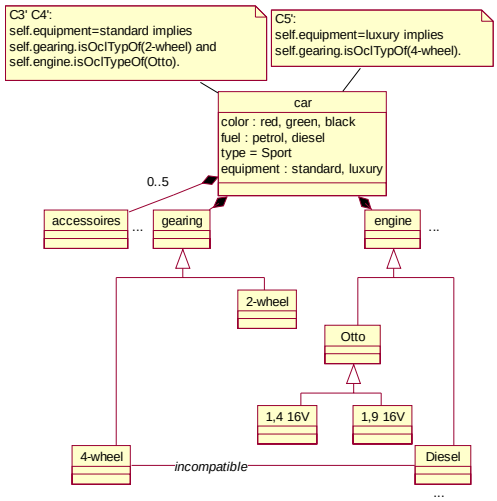


**Figure 7: car in context model „Sport“.**

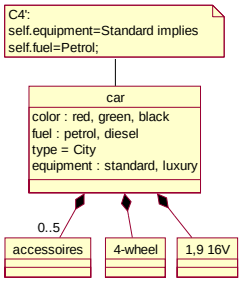


**Figure 8: car in context model „City“.**

## 5 MODEL BUILDING PROCESS

How should we proceed when modeling a configuration problem using the described concepts? First a root model must be constructed. After that one can repeatedly choose a model (in the first case the root model must be selected), select components out of this model (the corresponding constraints, cardinalities and attributes are inherited) and refine the structure of these components.

For the derivation of executable knowledge bases the semantics for the individual modeling concepts employed have been defined using transformation rules to a logic theory [2]. In the case of constraints that have to hold in some specific context the transformation rules into the logic representation are easily defined as follows.

Each context is characterised through a set of preconditions, e.g., some specialisation of parts according to the type hierarchy, a restriction of the multiplicities in an aggregational association, or the restriction of the domain of a property of an component type. These proconditions are basically a set of constraints. In the example of Figure 5, the precondition is that the car type is "Baseline". If more complex conditions are needed, the individual constraints for the preconditions are. logically conjuncted.

After the definition of the preconditions for a context, the designer has to define the constraints valid for that context, which are again some logical conjuncted expressions. A constraint is generated and inserted into the knowledge base. It is represented by a logical formula which has the following structure.

If {prec1, ..., prec-n} are the preconditions of a context in the graphical model (given as logical expressions ) and {constr-1, ... , constr-n} are logical expressions that have to hold in that context, then the following constraint has to be added to the knowledge base.

{prec-1 ∧ prec-2 ∧ ,..., prec-n}
{constr-1 ∧ constr-2 ∧ ,..., constr-n}.

Note that - given some specific context for the constraint - one can not define new component types, generalizations,

or aggregations. The more general product model can only be made more restrictive.

Example: Translation of the Baseline context into a logical representation using the notation introduced in Section 2:

type (X, car) ∧ val(X, type, Baseline) ∧ type(Y, gearing) ∧ conn(X, gearing, Y, car) type (Y, automatic).

type (X, car) ∧ val(X, type, Baseline) ({Y | type(Y, accessoires) ∧ conn(Y, car, X, accessoires)}) <= 2.

Note, that the generated constraints belong to one single model. The structuring on the conceptual level is not regarded at logical level where all constraints are integrated in one single knowledge base.

Each specialized concept model inherits all constraints and components of the more general model. The automatic generation of the knowledge base is done by traversing the derived inheritance hierarchy and adding constraints generated from the refinement operations.

## 6 CONCLUSIONS AND RELATED WORK

Research dealing with knowledge representation in the configuration domain regards the declarative constraint representation as the basic representation formalism since the development and maintenance of rule-based systems like R1/XCON [8] have shown to be error-prone. Using declarative constraint representation is not enough since knowledge bases become more and more complex so that conventional structuring mechanisms do not suffice.

Conventional mechanisms for structuring knowledge-bases in the configuration domain are described in Sabin, Freuder [12] or Peltonen et. al. [9]. There are many other works dealing with structuring the configuration knowledge. The common point is usage of conventional structuring mechanisms like abstraction, generalization and aggregation.

Feldkamp et. al. [1] discuss the structuring of the configuration knowledge in terms of reuse, i.e. they describe three different tasks when modeling a configurable product. First the properties of a product and the corresponding subsystems are modeled. Second the product structure is modeled by using the well known concept of partonomy. The third step is modeling the design logic behind the product. For resolving the problem of defining complex constraints over the product structure they introduce the concept of interfaces. In this context complex constraints are not defined in the components but are part of the interface which describes the connection between several complex components.

Haag [6] describes the structure of the SAP-configuration engine. Configuration in SAP means the creation of a concrete BOM conforming the customer requirements. This concrete BOM is created by pruning a so called maximum BOM, which is the counterpart of the maximum part structure presented in this paper.

Currently we have implemented a prototype for building UML-models (in Rational Rose) and do automatic translation into different logical representations of configuration tools. We are working on extending the prototypeto include the structuring concepts described in this paper.

## REFERENCES


1. F. Feldkamp, M. Heinrich, K.D. Meyer Gramann, "SyDeR System Design for Reusability" in AIEDAM Artificial Intelligence for Engineering Design, Analysis and Manufacturing, William P Birmingham ed., vol.12, no.4, sep 1998.
2. A. Felfernig, G.E. Friedrich, D. Jannach, "UML as domain specific language for the construction of knowledge based configuration systems" in proceedings of the 11th int. Conference on software engineering & knowledge engineering (SEKE '99).
3. A. Felfernig, G.E. Friedrich, D. Jannach, M. Stumptner, "Consistency Based Diagnosis of Configuration Knowledge-Bases" in proceedings of the AAAI '99 workshop on configuration, AAAI Press, 1999.
4. G. Fleischanderl, A. Haselböck, Thoughts on partitioning large scale configuration problems, AAAI-96 Fall Symposium "Configuration", Cambridge, Massachusetts, 1996.
5. G. Fleischanderl, G. E. Friedrich, A. Haselböck, H. Schreiner, M. Stumptner, "Configuring Large Systems Using Generative Constraint satisfaction" in IEEE Intelligent Systems, Configuration - Getting it right, jul./aug. 1998.
6. A. Haag, "Sales configuration in Business Processes" in IEEE Intelligent Systems, Configuration - Getting it right, jul./aug. 1998.
7. U. John, U. Geske, "Reconfiguration of Technical Products Using ConBaCon" in proceedings of the AAAI '99 workshop on configuration, AAAI Press 1999.
8. J. McDermott, "A rule-Based Configurator of Computer Systems" in Artificial Intelligence, Vol. 19, No. 1, 1982.

9. H. Peltonen, T. Männistö, T. Soininen, J. Tiihonen, A. Martio, R. Sulonen, "Concepts for Modeling Configurable Products" in proceedings of European Conference on Product Data Technology 1998, Sandhurst, UK, 1998.

10. B. J. Pine II, B. Victor, and A. C. Boynton, Making Mass Customization Work, Harvard Business Review, Sep./Oct. 1993, 1993

11. J. Rumbaugh, I. Jacobson, and G. Booch, The Unified Modeling Language Reference Manual, Addison-Wesley 1998.

12. D. Sabin, E.C. Freuder, "Configuration as Composite Constraint Satisfaction" in Artificial Intelligence and Manufacturing Research Planning Workshop, AAAI Press, 1999.