Chapter 10

Summary

This dissertation has tackled the problem of managing a variety of products that change over time. Such a variety of products is typically required for being able to control a specific market segment. Technical advancements, new designs, increasing customer demands, etc. drive the need to continually evolve the variety of products. In order to realize such a variety, a set of configurable products offer alternative and optional choices from which a customer can choose. To avoid time-consuming redesign and manual adaption, such complex products are assembled from a given set of smaller components. Configuration tools are widely used to reach this goal.

Configuration is a well-known approach to support the composition of products from a given set of components. Structure-based configuration, in particular, employs hierarchical specialization and composition structures in a configuration model. Within a configuration model, all potentially configurable products of a domain are implicitly represented by defining the components from which a product can be composed, component attributes and relations between the components. Similar products are configured from a common set of components while diversity is realized with different compositions, with respect to technical possibilities and given customer requirements. Structure-based configuration models are especially well-suited to represent products that are assembled from a given set of smaller components.

We call the continuous change of stored knowledge representing configurable products and the components from which products can be assembled evolution of configurable products. This dissertation has developed a knowledge management framework that supports the evolution of configurable products, enabling developers to concentrate on the product domain rather than on details of the underlying knowledge representation formalism. The developed framework includes representation and change management for all required configuration knowledge, i.e. both the components from which products can be assembled and the configurable products themselves, evaluation of impacts that changing the component base has on configurable products, and vice versa, evaluation of
impacts that changing configurable products has on the component base.

The knowledge management framework defines an intuitive and product-modeling centered set of modeling facilities. The modeling facilities concept, attribute, composition relation and constraint are mapped to corresponding elements of the knowledge representation language SWRL-\( \text{ALCQI}+\langle D \rangle \), which is a hybrid formalism integrating rules and Description Logics. The mapping provides a formal foundation for product modeling. The three change types addition, modification and removal together with the modeling facilities provide the set of elementary changes. For every elementary change, a base operation defines its formal semantics with preconditions and postconditions. The preconditions for a change operation encode what the model must be like in order for the change to be executable. The postconditions describe the immediate consequences resulting from the change. More complex changes can be defined through compound operations, forming a logical unity that is compiled from a set of base operations. While database systems typically decline changes that cause inconsistency, we rather allow the change and suggest suitable repair options. For every type of inconsistency there is a set of potential repair operations from which the developer can choose, leading to a “click-and-repair” process.

The effectiveness of a configuration application heavily depends on the quality of the underlying knowledge, that is, the configuration model. The term consistency and related terms are defined with respect to configuration models, together with basic inference services of Description Logics, such as satisfiability, subsumption, equivalence, disjointness and classification. Two more advanced inference services are developed especially for this dissertation: the invariant-based consistency checking and the optimized constraint satisfaction:

**Invariant-based consistency checking** consists of a set of invariants and a process to check consistency of a configuration model. The invariants define consistency of a configuration model. Invariants are formulae that have a particular status: they must be guaranteed to hold at every quiescent state of the model, that is, before and after a change is executed. Checking an invariant has one of two possible outcomes: it is satisfied or it is violated. A satisfied invariant denotes consistency with respect to this invariant while an violated invariant denotes inconsistency. A distinction between base invariants and extension invariants enables managing configuration models that are based on different language specifications. While the base invariants define a common basis of all language specifications, different combinations of extension invariants allow to check consistency of configuration models that rely on different, probably domain-specific or configuration-tool specific language specifications.

**Optimized constraint satisfaction** consists of a standard process for solving constraints and a set of optimizations that can be exploited for the conceptual representation of typical product domains. The optimizations are node consistency as preprocessor, which evaluates unary constraints first in order to eliminate local inconsistencies, reducing the search space, which propagates property values from leaf concepts to their parents in
order to prune unnecessary values and to reduce the search space for constraint satisfaction, \textit{evaluating constraints on the conceptual level}, which evaluates the constraints for instances of those concepts for which they are defines, not for every potential combination of instances of leaf concepts, and \textit{independent constraint subnets}, which evaluates constraints only when they belong to an affected constraint subnets, not constraints that belong to other, independent constraint subnets.

Knowledge management supporting the evolution of configurable products addresses managing both the representation of components from which products can be assembled and the representation of the configurable products. Both components and configurable products have their own evolution spaces. Hence, the following three scenarios emerge. All potential situations emerging because of managing changes to the stored knowledge are grouped into typical use cases in which a product manufacturer may be interested.

1. Managing the component base reflects changing the set of components from which products can actually be assembled while preserving consistency of the component model. Changing a component base does not only affect its own consistency but implicitly also affects the set of potentially configurable products. The following four use cases can be distinguished:

   - Defining a component.
   - Retiring a component.
   - Changing a component.
   - Managing constraints.

The use cases expose the need to evaluate the following technical questions:

   - \textit{Which configurable products are affected by a change?}
   - \textit{Which new components become available?}
   - \textit{Which components are no longer available?}
   - \textit{Can a configurable product still be configured?}

2. Managing configurable products addresses defining a variety of products that best suit the customer’s interests and reacting to influences of the market. Changing configurable products also concerns ensuring that the products remain configurable. A product is configurable if, and only if, its representation is consistent and covered by the component base. This means that all components from which a configurable product can be assembled need to refer to existing components of the component base. The following three use cases can be distinguished:

   - Defining a configurable product.
   - Retiring a configurable product.
   - Changing a configurable product.
The use cases expose the need to evaluate the following technical questions:

- Which new components are required to realize a configurable product?
- Which components are not required for any configurable product?

3. But also without explicitly taking evolution into account, there is some interaction between managing the component base and managing configurable products. The following four use cases can be distinguished:

- Creating a component model from a set of configurable products.
- Identifying required components.
- Checking satisfiability of components.
- Influencing the evolution of configurable products.

The use cases expose the need to evaluate the following technical questions:

- Which components are required for a configurable product?
- Which components and constraints are related to a configurable product?
- Which attribute values are consistent?
- Which property values are common for a set of configurable products?

This dissertation describes all use cases on a high level with examples while re-occurring reasoning services are defined in technical questions with formally defined algorithms, examples and an analysis of their computational complexity. All algorithms have exponential or better worst-case computational complexity, except for those use cases that include constraint satisfaction. Constraint satisfaction problems are known to be NP-complete. But the optimized constraint satisfaction focuses on aspects that – in most cases – considerably reduce computational complexity.

**Major Contributions**

Summing up, the major contributions of this dissertation are the following.

- Formal representation and reasoning support for all product-configuration related knowledge.
  - The dissertation introduces a semantic distinction between concepts that represent components (in a component model) and concepts that represent configurable products (in product models). This distinction stems from the different purposes of the representation. While basic definitions like satisfiability, subsumption or equivalence are the same for both kinds of representation, a configurable product needs to be covered by the component base.
  - Advanced reasoning services: besides the standard Description Logics inferences, this dissertation defines invariant-based consistency checking and optimized constraint satisfaction reasoning services.
* Both services are designed to especially exploit the peculiarities of representations of typical product domains.
* Notably, invariant-based consistency checking enables a domain-independent and configuration-tool independent way to check consistency of configuration models.

- The knowledge management framework introduces consistency-preserving change management that covers all possible situations, including:
  - Support for all potential changes to stored knowledge that is based on the defined knowledge representation language.
  - Evaluation of impacts that executing all potential changes to stored knowledge have:
    * Evaluating impacts that changes to the component base have on configurable products.
    * Evaluating impacts that changes to a configurable product have on the component base.
  - Analyzing impacts of separately managing a component base and managing configurable products.

**Proof of Concept** A prototype model editor has been implemented. The model editor allows creating and managing configuration models and evaluates impacts that changes to the component base have on configurable products. Conducted experiments demonstrate feasibility of the conceptual idea and show its scalability: executing change operations to a component model and evaluating impacts on configurable products are suitably efficient and practical also when applied to large configuration models. Additionally, the improvement of implementing the optimized constraint satisfaction is shown, compared to a standard generate-and-test algorithm.