Chapter 17
Variability Management

Georg Rock, Karsten Theis and Patrick Wischnewski

Abstract The global market, different and changing environmental laws, the customer wish for individualization, time-to-market, product costs, and the pressure on manufacturers to discover new product niches, to name only a few variability drivers, result in an ever increasing number of product variants in nearly all engineering disciplines as, for example, in car manufacturing. Mastering the related increasing product complexity throughout the whole product lifecycle is and remains one of the key advantages in competition for the future. Currently for a manufacturer, as for any other discipline, it is no option not to invest in an efficient and effective variability handling machinery able to cope with the arising challenges. Not only the task to invent, develop, introduce and manage new variants is important but also to decide which variant to develop, which to remove and which to not develop at all. The consequences of such decisions with respect to product-line variability have to be computed based on formalized bases such that an optimized product variability can assure on the one hand customer satisfaction and on the other hand cost reduction within the variability-related engineering processes. This chapter presents current research in the field of product variability configuration, analysis and visualisation. It presents solution sketches based on formal logic that were illustrated by some real world examples.

Keywords Product variety · Mass customisation · Product configuration · Complexity management · Variability management · Formal variability analysis

G. Rock (✉)
Trier University of Applied Sciences, 54293 Trier, Germany
e-mail: g.rock@hochschule-trier.de

K. Theis
PROSTEP AG, Dolivostr. 11, 64293 Darmstadt, Germany
e-mail: karsten.theis@prostep.com

P. Wischnewski
Logic4Business GmbH, 66123 Saarbrücken, Germany
e-mail: patrick.wischnewski@logic4business.com

J. Stjepandić et al. (eds.), *Concurrent Engineering in the 21st Century*,
DOI 10.1007/978-3-319-13776-6_17

17.1 Introduction

Product variety arises because customer requirements are usually individual and therefore they expect an individual solution for a given investment. Private consumers rank variety of assortment straight after location and price when naming reasons why they shop at their favorite stores. They care about variety because they are more likely to find what they want when going to a store that offers more varied assortments. In a market which is predominantly determined by the customer ("buyer market") and subject to high volatility, the variety of products for reasons of competition is presumably imperative because every manufacturer is forced to continuously extend the product portfolio with new, more efficient, more attractive products. Furthermore, the product variety occurs on the long-lived capital-intensive goods too, which were brought on the market to an earlier time with a lower level of modularity and now need to be overhauled or upgraded.

Thus, customers get a wide product portfolio offered, which allows them to find a product that best meets their requirements. The challenge lies in the selection of products, less the actual product offering [1]. Consumer demands have increased significantly in recent years and will increase again. Enterprises become successful if they can provide solutions for a variety of needs. The need to offer a variety of products marks the crucial difference with the past. From the marketing perspective, vendors offer product variety presuming that the product characteristics determine their value and that variety is a key driver of utility. Their assumption is that customers derive utility by choosing among singular characteristics of a product. Product variety can be just unnecessary, if the vendor is not able to meet customer's needs producing the most appropriate product from an existing modular product platform (see Chap. 14) [2, 3].

The use of information and communication technology in modern cars gives the product managers an even increased and thus arbitrary range of variety [4]. Especially the entertainment systems in automotive industry will be revolutionized concerning functionality and usability aspects. The seamless integration of mobile systems and even the utilization of mobile platform technologies within the car's entertainment system itself will result in new variants that have to be handled correctly in the car manufacturing context.

In a future perspective for the automotive industry, this trend of increasing variability is expected to be reinforced [1]. The automotive industry will meet a world of consumers in the future who want to get their individual wishes, needs, expectations and preferences expressed by their automobile and simultaneously the customer expects an improved product quality. Their individuality comprises various sub-areas such as mobility, urbanity, emotions, entertainment, security or knowledge. All these aspects can be condensed into the forecast that the passenger car in the future will be an expression of one's personality even more than today. It must, therefore, be better adjusted to the individuals than ever before [5].

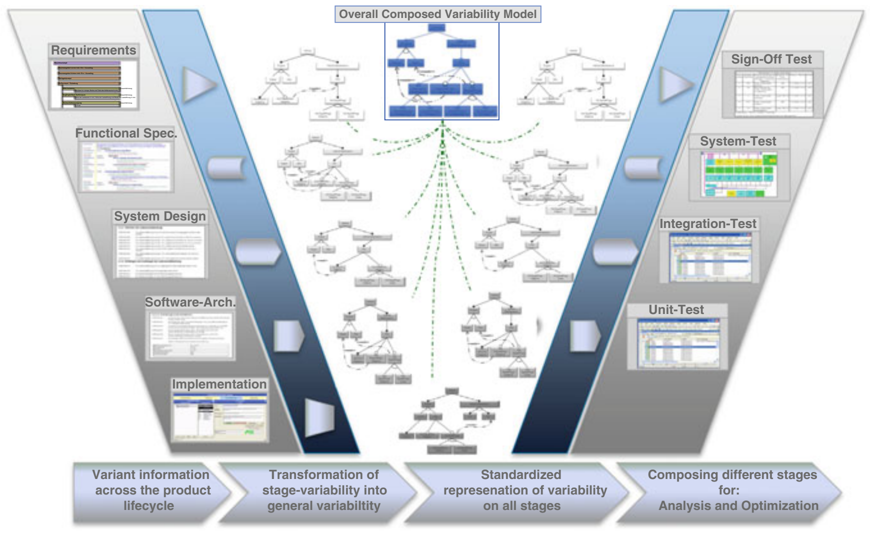

Overall Composed Variability Model
Requirements
Functional Spec.
System Design
Software-Arch.
Implementation
Sign-Off Test
System-Test
Integration-Test
Unit-Test
Variant information across the product lifecycle
Transformation of stage-variability into general variabiltity
Standardized represenation of variability on all stages
Composing different stages for: Analysis and Optimization

Fig. 17.1 Variability as a cross-domain and cross-functional development characteristic

Changing the view from a user of products to a developer of variant products and focusing on a V-Model based development the variability problem arises in every development stage as illustrated in Fig. 17.1 (see also Chap. 9).

Starting at the very early phase as for example in requirements engineering (see Chap. 5) variability constraints arise in all successive phases, as there are design, construction, implementation, and during the complete validation or verification phase. This shows two main aspects of variability in development (so-called internal variability). First, variability is not limited to exactly one development phase, but it rather spreads over all development phases. Second, since there are dependencies between domains and corresponding functions we have to have an "overall composed variability model" as depicted in Fig. 17.1. This model tends to be very large and complex. Thus, there must be a very powerful methodology and corresponding analysis engines supporting the engineers during product development.

The structure of this chapter is as follows. In Sect. 17.2 we present some background information including a short introduction to the current research in variability management. Section 17.3 changes the perspective from a user of variable products to the developer of products with variability. The related development process, challenges and corresponding concepts for a tool-based solution are presented. In the following Sect. 17.4 the application of the described concepts is shown with the help of industrial examples. Although these examples are rather abstract they illustrate typical application scenarios. Section 17.5 presents use cases. Two final Sects. (17.6 and 17.7) sum up and give some future directions in the field of variability management and variability analysis.

17.2 Background

Product variety encompasses different product designs or product types that are offered in a market by a vendor. It can be subdivided into external variety and internal variety. External variety is the product variety seen and perceived by the customer, whereas internal variety is the variety covering all procedural variants inside manufacturing, such as logistics and distribution operations in satisfying the provision of external variety. External variety is further subdivided into useful variety and useless variety. Useful variety is appreciated by the customer, such as useful options and stylistic differences, in distinction to the useless variety that is either transparent or unimportant, or confuses the customer. Uncontrolled internal variety may yield the excessive and unnecessary variety of parts, features, tools, fixtures, raw materials, and processes. Although more product variety extends the potential for generating increased revenues, there are potential adverse effects, resulting from increased complexity.

External variety is an important driver for enterprises producing to forecast. On the other hand management of internal variety is dominant for enterprises offering products to order. The effectiveness of strategies to mitigate negative effects of internal variety, such as modularity, mutability, late configuration (postponement), and option bundling, depends on the order-fulfillment strategy the enterprise follows [1]. This requires that assembly systems have to be designed such that they can handle the variety, too [6–8].

Hence, the main objective of many recent research projects is the decrease of external variety by standardized interfaces and further module utilization in different combinations and simultaneously the limitation of internal variety by standardization and modular design. In addition, research has focused on the identification of the right degree of variety [9–11]. This aims at reducing the complexity such that it can be handled with current tools [11].

Now, the question arises how the remaining internal product variety can be managed such that the following requirements are simultaneously fulfilled:

- As many customer requirements as possible are fulfilled,
- the quality of the products is improved,
- the development and production time is shortened, and
- the overall costs are reduced.

Since product variety has gained a high importance in the past years, a new, autonomous cross-functional discipline called variety, variant, or variability management was established for holistic treatment of these phenomena. In particular, powerful tools that shift dealing with the complexity from the user to the tool, is, in our opinion, the key for successfully achieving the above requirements.

The feature oriented domain analysis-approach (FODA) introduced by Kang [12] uses feature models which are today almost established as a standard mean to specify the variability of products. They allow for a description of the domain engineering and the application engineering model variability within the Product-Line-Engineering

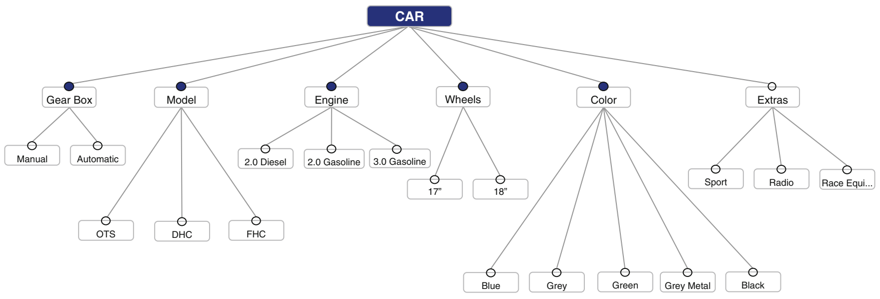


Fig. 17.2 Product structure

process as described in [13]. Although feature models have not yet pervaded the complete market for variant reach products, we observed over the last years that companies using their proprietary formats and analyzing techniques (often based on Excel-like tools) reach the limits of their approaches and are looking for a more sophisticated analyzing machinery and a standardized form of variability specification as started for example by the Object Management Group (OMG) Request for Proposal for common variability language (CVL) [14]. Feature models enable us to visualize variability in multiple ways as shown in [15–19]. All the referenced approaches are using tree like views as for example shown in Fig. 17.2. Furthermore they give us the possibility to analyze the specified variability in a formal way because of the defined and generally accepted formal semantics as described for example in [20–22]. There are several extensions of these models including the handling of additional feature attributes to be part of the formal analysis process as described in Sect. 17.3. At least from an academic point of view variability related problems in product development seem to be solved and do not need further investigations [23]. A closer look at real world applications reveals that many of the problems remain still unsolved for different reasons. A short and by far not complete list of reasons is as follows:

- The pure size (still increasing) of real world problems,
- Missing formal variability experts able to improve formal analysis algorithms on a problem/product/enterprise based way,
- Grown product and management structures to handle variability established in the last decade,
- Missing migration concepts for established and grown legacy systems to handle product variability,
- Increasing product and environment complexity,
- A constantly increasing number of variability drivers.

The only tool able to solve the problem is the formal logic (with different characteristics). Today, most of the variability tool vendors use operational logic that evaluates the rules in a particular order. This approach usually has unwanted side effects because different execution orders can lead to different results. As a consequence the encoding of variability in an operational logic is hard to debug. Furthermore, these tools are usually not complete which means that the tools do not consider all possible variants. These tools are already of big help, but will not succeed to solve the problem in an adequate way.

In order to obtain complete tools and avoid side effects, all rules have to be considered at once without any order. There are so called formal solvers that accomplish these requirements. However, the methods that these solvers implement are computationally expensive. This means that they are not feasible for industrial size problems, in general. As a consequence, the methods as well as their implementation in solvers have to be adapted to the respective input in order to obtain efficient solvers for dealing with the variability in industry. Although general purpose solvers can be applied to many industrial problems with some success, it is in our opinion not enough to be able to solve say 80 % of the problems and leave the rest with the so-called "local heroes".

In addition, a solver alone does not solve the variability problems in industry because there is a strong need for a visualization that represents the solver results in an adequate way to the user. The visualization of product line models and their dependencies that represents the far end in product line engineering has to deal with huge variability models and often an incredible number of constraints (more than 200,000) for a complete product line. To the best knowledge of the authors there is currently no tool with an appropriate user interface available able to visualize such models, their constraints and analysis results from a solver in an effective and efficient way.

In order, to obtain such a comprehensive tool that efficiently analyzes product data and visualizes the models and the analysis results, requires two directions of research. One direction focuses on the development of reasoning procedures that efficiently analyze product data. The second direction of research focuses on the development of an appropriate user interface. With successful research in these areas, the product life cycle management would be revolutionized and there would be a significantly improving of product quality accompanied by a reduction of the development time and development costs.

This chapter presents current research towards this goal based on formal logic. Propositional logic has been recognized throughout the centuries as one of the corner stones of reasoning in philosophy and mathematics. With the help of proposition logic and sometime needed slight extensions to it, a wide range of combinatorial problems arising in variability management can be expressed and formulated as a propositional satisfiability (SAT) problem (see Sect. 17.3.2). Meanwhile SAT has become a reference problem for an enormous variety of complexity statements [24]. The presented method is based on SAT and its extensions that have been successfully used by the authors in several industrial projects. In addition, this chapter shows extensions of these methods and presents examples where these methods have been successfully applied. In addition, several directions for further research towards a comprehensive tool suit are sketched.

17.3 Functionalities for Management of Complex Products

In the concurrent engineering process, it is vital to have comprehensive analysis tools that ensure properties like consistency, correctness and ensure that specified properties are fulfilled. This is required throughout the whole life-cycle of the products. In particular, complex products can be composed of several thousand parts. In this context, complete analysis tools are indispensable that help the engineers to manage the product data, significantly improve product quality and reduce overall cost.

These tools identify errors in the development process as soon as they occur and can, therefore, be identified and avoided from the very beginning. This saves the possibly extremely expensive correction of these errors in a later stage of the development process or even after the product has been delivered to customers. The possibility to personalize a product in terms of individual customer preferences is nowadays one of the key factors in many industries.

Not only the presentation of variety but also the effective and efficient management of variety come into play. To address the mentioned challenges, there is a need for methodologies to specify, analyze and manage the occurring variety efficiently. In this chapter, we concentrate on the correct and complete analysis of industrial-sized variant descriptions. This is the computationally most difficult part moving the handling of complexity from the user to the algorithm. So, it is a powerful tool for the user, in order to benefit from the complexity of the products with respect to variety, quality and efficiency.

Below, examples are presented of an analysis task that a respective tool should be able to perform in order to successfully and completely manage all variants of a product:

- Detect inconsistencies and conflicts in the product variability description
- Determine all parts of a specific product
- Determine all parts that are currently not used in any product
- Determine in which products a particular part is used
- Compute product configurations with predefined properties, i.e. buildability
- Computation of the needed (optimal) variability for a product family
- Optimization of a product family according to predefined measures such as cost or customer orientation
- Identify all reusable components
- Identify products that are not desired by the costumers.

With such a tool it is not necessary anymore to restrict the number of variants as much as possible in order to maintain them. Moreover, it allows a manufacturer to maintain many more variants of a product by reducing the effort to manage all of them.

A tool providing this functionality must be functionally correct, complete, fast and efficient. In particular, it has to consider all possible product variants during the analysis. Approximations are not suitable for this purpose and must be excluded in this phase, because they do not consider all possible variants, i.e. they omit products during the analysis of a product portfolio. Thus, they do not guarantee to find all relevant information or errors in the product data. We argue that an incomplete approach is not good enough and, furthermore, can mislead and direct wrong decisions.

Because of the fact that discrete products have a discrete structure, complete methods are computationally expensive. In contrast to a continuous structure, a discrete structure means that a product is composed of several individual parts. Each part is either contained in the product or it is not contained. In general, complete procedures for discrete structures are at least NP-hard. For the worst case, this means that all combinations of all parts of a product have to be considered during the analysis. For example, for a product portfolio with 3,000 parts we have, in general, $2^{3,000}$ different products that have to be considered during the analysis. The same holds in case of engineering changes, which must be considered too.

To obtain efficient and complete analysis procedures for complex products, the encoding of the product data and the reasoning procedures have to match to one another. The research in the field of automatic theorem proving of the last decade has improved the underlying algorithms in such a way that product data containing several million product parts can be analyzed efficiently in many cases. In the remaining of this section, we present a complete method for the analysis of product data based on a particular kind of logical reasoning procedure, based on SAT [24] that was successfully applied under industrial conditions.

Figure 17.3 depicts the general workflow of analyzing and optimizing products according to the presented method. First the product data has to be transformed into a logical model representing all the relevant aspects of the product data (see Sect. 17.3.1). In the second step, the logical reasoning procedure that analyses and optimizes the logical product model (see Sect. 17.3.2) is applied. The result shown to the user depends on the performed analysis. In nearly all cases the result has to be adapted to the expected reader to be understandable and, thus, usable.

17.3.1 Logical Product Model

Transforming the product data into a representation in logic (as depicted in Fig. 17.3) defines a unique semantics for the data. Thus, logical reasoning tools can precisely analyze the logical model. In order to obtain an efficient and useful analyzing framework that supports the concurrent engineering process, the logical product model has to fulfill the following three substantial requirements:

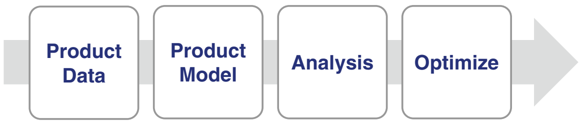


Fig. 17.3 Analysis process

- The model must adequately represent the relevant information of the original product data.
- The model must be in a form such that efficient analyzing procedures cope with model complexity.
- The structural representation of the model allows to trace back the results of the analysis to actual product characteristics.

Feature diagrams [25, 43] are an appropriate means, because they provide a logical framework that fulfills these requirements. They are suited to encode structural information of a product as well as the formulation of additional constraints.

This is an important step in order to build a knowledge base for all products of a manufacturer. In particular, this knowledge base is a tool that enables the engineers to consolidate their knowledge into one consistent knowledge base. The reasoning procedures shown in Sect. 17.3.2 provide a tool that automatically verifies the consistency of structural information and additional constraints, and prevents possible interface problems between several engineering departments. As a consequence, this knowledge base is an explicit representation of the manufacturer's know-how and enables everybody involved in the product lifecycle to access this knowledge (see Sect. 17.4.1).

Besides the mentioned advantages it should be noted that the creation of an appropriate logical representation of the product model is usually not a trivial task. It needs a lot of experience in the field of formal logic and their respective analyzing procedures. Furthermore, a gap remains between the real world product and the specified formal model. This gap can only be bridged with the help of the domain experts who should provide a crucial support within this first phase.

17.3.1.1 Feature Diagrams

Figure 17.4 depicts a feature diagram that represents an extension of the product structure given in Fig. 17.2. A feature diagram is a tree that represents a set of valid products which are also called variants. Thus, variants represent a valid selection of features (nodes in the feature tree) in a feature tree. It is possible to further restrict valid feature selections by so-called cross-tree constraints.

- Optional/mandatory features
  In every valid product a mandatory features (indicated by a filled circle in Fig. 17.4) is selected if and only if its parent node is selected. If an optional feature is selected, its parent has to be selected, too. The root of the feature diagram is usually mandatory and assumed to be part of all valid products.
- XOR, OR, AND group
  If a non-leaf feature F is set to XOR, exactly one of its children has to be selected if F itself is selected. In Fig. 17.4 the feature Engine is a XOR group, i.e. the car can have exactly one engine and in this case must have exactly one

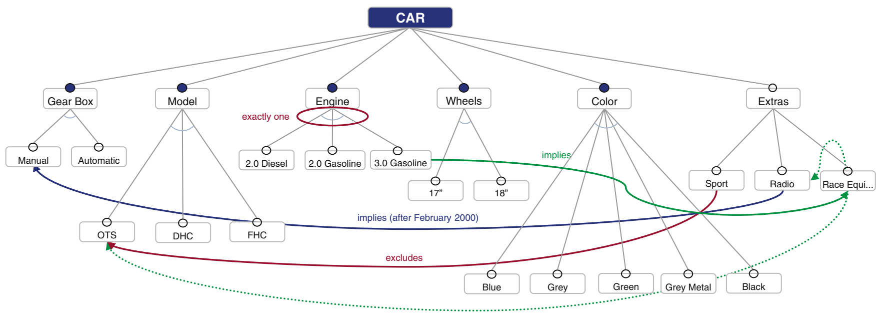


Fig. 17.4 Feature diagram

engine. Likewise, OR denotes that at most one children is selected and AND that all children have to be selected.

- Implies/Excludes
  The implies and excludes relations define so-called cross-tree constraints. For example, if the feature "3.0 gasoline" is selected then the feature "Race" has to be selected and if the feature "Radio" is selected then the feature open twoseater (OTS) is excluded, i.e. it cannot be selected.

Additionally, a feature diagram can contain arbitrary Boolean formulas over the features in the tree. The following formulas are examples for such constraints:

$$\text{Race} \rightarrow \text{Sport} \bigwedge \text{OTS}$$

This means whenever the Race option is part of a product then also the Sport option and the option OTS have to be included in the product.

The second example expresses the constraint that the OTS excludes a radio:

$$\text{OTS} \rightarrow \neg\text{Radio}$$

Feature diagrams represent product data with a precise semantics. Consequently, a feature diagram defines a mathematical model for the variability of the corresponding product data. This model constitutes the basis for the formal analysis described in Sect. 17.3.4.

17.3.1.2 Quantitative Extensions of Feature Diagrams

An extension of feature trees by attributes as mentioned by Benavides [26] extends the expressivity of feature models (see Fig. 17.5). Attributes provide a method to model additional information within the feature model. With appropriate reasoning methods these attributes can be used in the analysis process and the visualization. Examples for such attributes are costs, weight, and speed. There exist reasoning procedures for analyzing attributed feature trees and for optimizing in terms of the specified attributes.

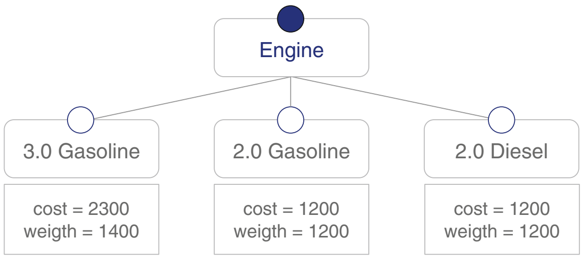


Fig. 17.5 Extended feature diagram

17.3.2 Logical Analysis

In order to perform logical operations on feature diagrams, the feature diagrams are first translated into a set of Boolean formulas. There exist tools that automatically verify the satisfiability of a set of Boolean formulas. The tools we consider here are so called SAT solvers [24].

All aforementioned analysis tasks can be formulated as a Boolean satisfiability problem and, consequently, can be answered by a SAT solver.

Aside from efficient SAT procedures, the right encoding of the real-world problem in Boolean logic, is the key for successfully analyzing industrial product data.

17.3.2.1 Translation into Logic

Feature diagrams are a formal representation of product data with a precise semantics. Consequently, they define a unique mathematical model of the product data that consists of the hierarchical structure and additional logical formulas that specify the properties and links to all product parts. The analyzing procedures presented here operate on purely logical representations of the product data. As a consequence, the structure and content of a feature diagram have to be transformed into an equivalent representation in logic.

The following example shows the translation of the node GearBox of Fig. 17.4 in propositional logic:

$$\begin{aligned}\text{GearBox} &\rightarrow \text{Manual} \vee \text{ManualCR}\\ \text{GearBox} &\rightarrow \neg\text{Manual} \vee \neg\text{ManualCR}\\ \text{Manual} &\rightarrow \text{GearBox}\\ \text{ManualCR} &\rightarrow \text{GearBox}\end{aligned}$$

This expresses the parent-child relation and the property that this node is an XOR node.

Although the translation of a feature model into a set of Boolean formulas is straightforward as proposed by Benavides et al. [26], this is a crucial step for successfully analyzing huge product data. Because they are computational expensive, the reasoning procedures rely on the right encoding of the input data in order to perform efficiently and to avoid the worst-case behavior.

From our experience, a respective encoding for most problems from industry can be found, which has also been proposed by Mendonca et al. [21]. However, this is not always obvious and might involve deeper inspection of the product data performed by the variability expert together with the domain expert.

17.3.2.2 Satisfiability Procedures for Feature Diagrams

The analysis tasks of Sect. 17.3.1.1 needs be transformed into a satisfiability problem. A satisfiability problem verifies for a set of Boolean formulas if there exists a variable assignment that fulfills all formulas at the same time.

Considering the following set of Boolean formulas where a, b and c are Boolean variables, i.e., they can be assigned true or false:

$$\begin{array}{ll} a \vee b \vee c & \neg a \vee b \vee \neg c \\ a \vee \neg b \vee c & a \vee \neg b \vee \neg c \\ a \vee b \vee \neg c & \neg a \vee b \vee c \\ \neg a \vee \neg b \vee c & \end{array}$$

Although this is a small problem, the solution is not obvious. If we consider problems with several thousand or million variables, this is almost impossible to do without efficient and sophisticated algorithms. In order to find a solution, one has to consider all possible assignments for the variables. Assuming a problem with 50,000 variables, we have to consider $2^{50,000}$ assignments. Once a fulfilling solution has been found, the solution is easy to verify. These kinds of problems are called NP-hard problems.

In 1960 an algorithm [27] was presented that computes an assignment for a set of Boolean functions. This algorithm was later improved and became known as the DPLL algorithm. The algorithm searches the decision tree as depicted in Fig. 17.6 for a satisfying solution.

In order to perform the satisfiability procedure efficiently, the DPLL algorithm has been improved by many techniques and methods. Tools that implement a satisfiability procedure are called SAT Solvers which are described in detail in the Handbook of Satisfiability [24]. Current SAT Solvers are mostly based on the

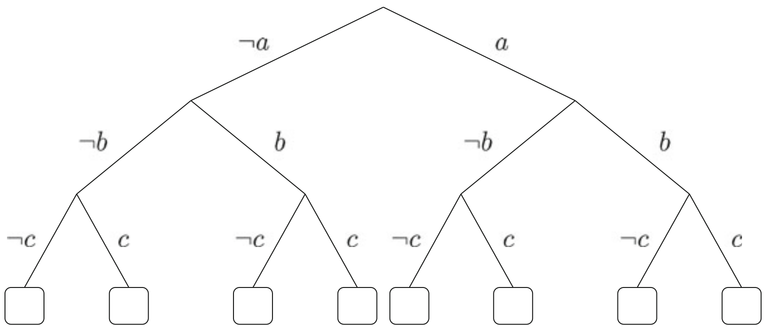


Fig. 17.6 Decision tree

Unit Propagation	$(M, N \cup \{C \vee L\}) \Rightarrow (M + L, N)$	$M \not\models C$ $L \notin M$ and $\neg L \notin M$	(1)
Decide	$(M, N \cup \{C \vee L\}) \Rightarrow (M + L^d, N)$	L occurs in N $L \notin M$ and $\neg L \notin M$	(2)
Fail	$(M, N \cup \{C\}) \Rightarrow \text{fail}$	M does not contain any decision literal $L \notin M$ and $\neg L \notin M$	(3)
Backjump	$(M' + L^d + M'', N) \Rightarrow (M' + L', N)$	$N \models C \vee L'$ $M' \not\models C$ $L' \notin M$ and $\neg L' \notin M$	(4)

Fig. 17.7 Abstract CDCL procedure

CDCL method that is an improvement of DPLL. The abstract DPLL calculus [44] is depicted in Fig. 17.7.

However, in general, it is not sufficient to use an off-the-shelf solver, because also the solver engine has to be adapted to the input problem in order to obtain the best, or in many cases an acceptable, performance. The harder the problems become, the more crucial it is to adapt the solver in order to solve the problem at all.

17.3.2.3 Reasoning Procedures for Extended Feature Diagrams

A feature diagram as well as an extended feature diagram has an equivalent representation as a set of logical formulas (see Sect. 17.3.2.1). The translation of an extended feature tree results in a propositional logic with an additional theory. For this kind of logic there exist three approaches of a reasoning procedure in basic research: modular [28–30], hierarchical [31, 32] and integrated approaches [33, 34].

The modular approach implements a black-box strategy whereas the other two approaches implement a white-box strategy:

(i) Modular approach: The modular approach adds the quantitative aspects as labels to the propositional formula.

$$\{\mathrm{weight} = 5, \mathrm{torque} = 450\}\colon \mathrm{ManualCR}$$

(ii) Integrated approach: The integrated approach combines the quantitative aspects with the propositional formulas resulting in one formalism.

$$\mathrm{ManualCR.data} = (5, 450)$$

(iii) Hierarchical approach: The hierarchical approach combines the quantitative aspects with the propositional formulas, too. In contrast to the integrated approach, the combination of these two formalism is only through variables.

$$x = 5 \wedge y = 450 \rightarrow \mathrm{ManualCR.data} = (x, y)$$

The modular approach results in two almost independent formalisms. This means for the analysis that the theory part is not integrated in the actual analysis. Rather, it is analyzed independently with special methods for the respective theory. As a consequence, this method is called black-box strategy because the analysis procedure does not know anything about the theory methods. It only asks the theory black-box if a given label is valid in the theory part of the formalism.

The two white-box approaches combine the propositional and the theory part within a single formalism. On the one hand, this has the disadvantage that the two parts cannot be treated independently from each other. This results in methods that are considerably more complex. On the other hand, the white-box approaches are transparent. This means that the properties of the theory part of the formalism can be easier used in the propositional part in order to simplify the current formulas.

Adding taxonomies to the logical formalism of complex products is a further extension that requires decision procedures for the Bernays–Schönfinkel class. This is also called the effective propositional (EPR) class. A decision procedure for the Bernays–Schönfinkel class based on the first-order prover SPASS [35] is presented in [36]. An overview over EPR solvers can be found in [37].

It requires further investigation which of these approaches is best suited for analyzing the respective properties of industrial products.

17.3.2.4 SAT Based Optimization

In addition to the analysis, there exist efficient optimization procedures for product models that compute optimal products with respect to a given cost function. Because of the fact that product data have a discrete structure, computing an optimum requires discrete optimization procedures. In general, discrete optimization problems are much harder than continuous optimization problems. For further details about optimization in general and further use cases, Chap. 15 may be considered.

The hardness of discrete optimization problems for huge products such as cars, aircrafts or ships involving several thousand parts, requires highly efficient optimization techniques.

We have successfully used SAT based optimization procedures in several industrial projects applying the branch and bound method [38]. Using this procedure together with the logical product model (see Sect. 17.5.1), a defined cost model and an objective function computes optimum products with respect to various cost metrics.

For example, the following use cases can be solved with such an optimization procedure:

- Assembly sequence optimization [6],
- Computation of the optimal configuration, for example the cheapest, fastest, lightest product (see Sect. 17.4.2).
- Test coverage optimization.

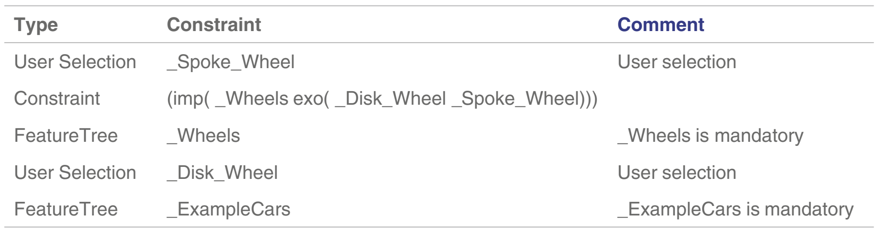
An extended feature model as depicted in Fig. 17.5 defines a product model and a cost model. In addition to the objective function, these methods allow the formulation of additional bounds.

17.3.3 Visualization

In addition, to visualize the product structure as depicted in Sect. 17.3.1, the analysis and optimization results need an adequate visualization. The visualization should provide the results to the target user group or role, for example engineers, managers and sales people.

This means that the visualization of the same analysis task has to be visualized with respect to the specific user scope, considering specific needs or objectives e.g. level of detail or abstraction. The reasoning procedures shown in Sect. 17.3.2, produce the respective information.

However, preparing and visualizing this information such that the user efficiently can use it is an open area for research. Figure 17.8 illustrates an example for a reporting of a conflict. In this example, a user has selected two different kinds of wheels, but only one is allowed by the constraints. In the case that a conflict is more complicated and involves hundreds of features and constraints, this representation is not appropriate anymore in order to efficiently analyze the conflict.



Type	Constraint	Comment
User Selection	_Spoke_Wheel	User selection
Constraint	(imp(_Wheels exo(_Disk_Wheel _Spoke_Wheel)))	
FeatureTree	_Wheels	_Wheels is mandatory
User Selection	_Disk_Wheel	User selection
FeatureTree	_ExampleCars	_ExampleCars is mandatory

Fig. 17.8 Conflict reason

17.3.4 A Tool for Formal Variability Management

The presented analysis methods and the presented examples described in Sect. 17.4 are realized with the help of a commercial tool [39]. As opposed to tools as for example described in [40, 41], this tool focuses on the analysis and optimization of variant product structures based on the aforementioned propositional logic and a very efficient implementation of a SAT solving procedure. It offers a set of analysis possibilities as there are for example:

- Consistency Check
- Dead Feature Detection
- Optimization with bounds
- Product Configuration
- Analysis Result Visualization
- Formal Rule-based Debugging.

Considering these analysis procedures the focus of this tool, lies in the performance of the result computation. The user is now able to solve the specified problems in real-time. The tool allows for a systematic and timesaving analysis of huge product structures, keeping at the same time the right level of abstraction for the result presentation. It can be looked at as a formal logic based integrated development environment (IDE) for variant product structures where all the analysis results are mathematically proven to be correct. There is no longer a “perhaps”.

17.4 Applications

This section presents examples showing how SAT based methods can be used in order to significantly improve the overall quality of complex product data and, consequently, the quality of the actual products. As a consequence, they enable manufactures to offer more variants that they can manage. This allows them to satisfy the customer’s requirements with higher granularity.

The examples shown in this section are a product consistency check and product optimization.

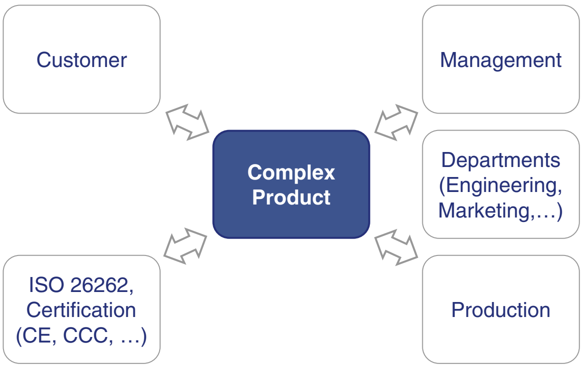

Customer
Management
Complex Product
Departments (Engineering, Marketing,…)
ISO 26262, Certification (CE, CCC, …)
Production

Fig. 17.9 Parties involved in the product lifecycle

17.4.1 Product Consistency

A complex product involves many parties, such as customers, mechanical engineering, electronic engineering, marketing, management, regulations and certifications. Ensuring that a product respects all requirements from all parties is a challenge that causes a lot of problems, in general. These problems are the major issue for product callbacks or severe damage to people and the manufacturers. Figure 17.9 illustrates the different parties that have particular requirements to a product.

Verifying that a product with several million variants satisfies all the requirements, is only possible with complete methods. Completeness means that all possible variants are respected and checked that they fulfill all specified requirements and satisfies all interfaces to all involved parties. This procedure is called a consistency check.

First, in order to use logical methods, the product model as well as the specifications must have been formally specified. Mostly, products and its specification are stored in a product data management (PDM) system. Consequently, these data have to be translated into a formal representation first.

In order to perform a consistency check for a product, all relevant aspects of the product that are not contained in the PDM system, have to be also present in a format that can be transferred to a formal model (see Sect. 17.3.1). For the estimation of costs, this could also involve enterprise resource planning (ERP) systems. Currently, there exists no general approach. Therefore, it is defined for every use case. This leaves room for research in order to find a general approach.

Even more, such a model improves the communication between the parties by defining a precise language that can automatically be verified against specified properties.

Once all the aspects are defined and transferred to a formal model, the formal methods of Sect. 17.3.2.2 verify their consistency. If there are inconsistencies, these methods will find them and generate an explanation. This explanation is called a proof. A proof is a mathematical precise representation of the reason of the inconsistency. Based on this proof, sophisticated and exact reports for each party involved can be generated together with proposals to solve them. This is an area for further applied research, too.

This section depicts the opportunities that formal methods provide in order to support the product lifecycle and to improve the product quality by ensuring that all requirements of all parties are fulfilled. At the same time, this section shows the areas for further research towards a general approach.

17.4.1.1 Industrial Application

The product model represents the information backbone for all people working on a specific product. In automotive industry for example, engineers must ensure that new parts fit into a well-defined set of car configurations—geometrically and logically. During the development phase the product structure and the geometrical parts are developed concurrently, thus both models are subject to a continuous change. In order to cope with the described situation, engineers work in so-called "reference configurations". These reference configurations have to provide a valid reference for all product configurations using the designed parts, functions etc. Changes in the variant product model may have various effects on this: The reference configuration may represent the wrong set of configurations, or an incomplete set or the configuration becomes invalid at all. Obviously, managing complex products within a complex project environment with concurrently working engineers, this happens each and every day. The work of all mechanical designers, electrical engineers, software developers etc. deeply depends on a consistent product model. No person can detect all errors in the variant product model manually. The risk is that errors propagate through follow-up processes undetected, causing very large efforts and high costs. By this an automatic efficient consistency check of variant product models becomes an essential success factor for all companies offering variant products.

17.4.2 Product Optimization

The optimization methods in Sect. 17.3.2.4 allow the computation of optimum complex products with respect to specified metrics. Examples for cost metrics are price, weight and CO_2 emission.

The following shows examples for optimization tasks:

- What is the best product with respect to customer needs?
- What parts of a company portfolio are cheaper to be produced in house and what parts are cheaper to be bought from a supplier?
- What are the parts that cause the most costs?
- What parts are worth to consider for redesign in order to make them more cost efficient?

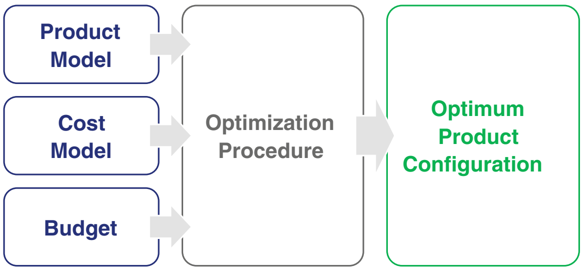


Fig. 17.10 Optimization of product configuration

Figure 17.10 depicts a black-box view to the optimization procedure for complex products. The inputs for the optimization procedure are a product model, a cost model and sales figures. The product model represents the product data and the cost model assigns certain cost values to the product model. The sales figure is a list of product variants that are expected to be sold or were sold in a defined time period.

For example, based on this input parameters the optimizer computes the cost-optimal product model by removing or adding parts and products to the product portfolio. The computed optimum is consistent in terms as explained in Sect. 17.4.1, i.e. all requirements are fulfilled. In addition to the consistency of the computed optimum, this method finds mathematically provable the global optimum.

The results of the optimization procedure can be the basis for further decision processes. They can optimize on several expected market situations and the respective consequences. In this case, the formal methods provide the presented functionality, but they have to be adapted to the individual application in order to perform efficiently. This is due to the fact, that an optimization operation is even more difficult from a computational point of view than a consistency check. Therefore, it is even more sensitive to the given product model. As a consequence, research towards a general approach for optimization of complex products is needed. Furthermore, there is currently no standard mechanism for the specification of the input parameters like the costs for the product model or the sales figures.

17.4.2.1 Industrial Application

Product variation is a key differentiator between competitors. However, product variation is causing costs in development, production and after sales too. The optimum is in-between the broadest offering to get more customers and the smallest offering to save costs. Real customer orders aren't evenly spread across the possible product configurations. It's the exact opposite: Real customer orders show accumulations of few sets of options, because there are always customer groups with similar requirements. One possible optimization is to offer packages with a well-defined set of options. These packages fulfill the requirements of many customers and reduce the number of variants and the costs too. However, the optimal set of packages is very hard to find, because no one can manually survey the mix of a complex cost structure, historical selling and forecast data within a highly variable product model. The identification of an optimal set of packages is the typical use case for a mathematical optimization toolkit. Even in this case the customer has to be taken more closely into account. Often the customer selects a package and afterwards wants to upgrade a certain feature within that respective package. Also in this case the optimization toolkit has to provide a useful variant.

17.5 Case Studies

In this section we present real industrial case studies for the application scenarios described in Sect. 17.4. These case studies were abstracted from the real case for obvious reasons.

17.5.1 Debugging a Product Variant Model

In this case study, a debugging tool for the product data of a globally operating machine manufacturer is presented. The debugging tool ensures that the product data fulfills the quality requirements. The manufacturer uses several mechanisms to express properties of his product involving engineering and marketing data. Altogether, the manufacturer defines several thousand of such product properties. At the engineering side the properties of the product describe all products that are buildable. The marketing side describes the market requirements and the product variants that are offered in the different countries.

The manufacturer uses a configuration tool that allowed the user to configure the product in the respective country. However, this process was error prone because the configuration tool did not offer a comprehensive analyzing mechanism, making the specification of the product model and the resulting variants transparent and understandable to the user.

The absence of such an analyzing mechanism results in a sometimes-unpredictable behavior of the configuration tool. Examining the specification mechanisms of the manufacturer in order to understand the reason for the behavior was very time consuming and required a lot of experience.

This has two reasons. First, the definition of the specification mechanisms was not unique and several exceptions to the rules were implemented. Second, the configuration engine used an operational semantics. This means the interpretation of the mechanisms was implemented in the software. As a result, the order of the rules had an impact on the behavior, which is usually an unwanted side effect.

In the case study, we used the commercial analyzing tool [39] that solves these problems by enabling und actively supporting the user to understand the problem

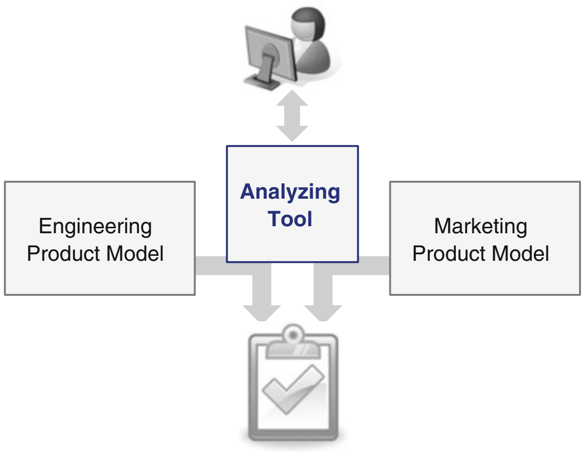


Fig. 17.11 Debugging process overview

and to find out whether the problem was a specification error or a bug in the configuration tool. Figure 17.11 depicts the debugging process and the respective inputs. For example, we discovered with the help of the tool that marketing offered a control unit for a variant of a machine for which the control software had not been released yet.

The first step in this investigation has been the translation of the specification mechanisms of the manufacturer into a logical product model. This defines a formal semantics based on mathematical logic for each individual mechanism and, as a consequence, each mechanism has a unique precisely specified meaning.

The logical product model can be loaded into the analyzing tool. The tool allows the user to interactively inspect the product model. Furthermore, the consequences of a selection of a feature in the product are computed in real time. Even more important, the user sees the reason for the consequences. In particular, the user immediately understands why the selection of one feature requires the selection/de-selection of other features.

In addition, the tool finds inconsistencies immediately and, therefore, avoids working with an incorrect product model. Usually a conflict involves just a few rules. As a consequence, the rules to be considered in order to understand the reason of an inconsistency is drastically reduced from several thousand to just a handful of rules.

Additionally, the tool enables even less experienced people to understand the specified product properties and its consequences. This is because the tool serves as a reference interpreter for the properties. As a consequence, the communication between several people involved in the product lifecycle of the product was significantly improved.

17.5.2 Computation of a Set of Cost-Optimized Wire Harnesses

In this case study the goal was to compute a set of cost optimum wire harnesses for a product portfolio of a manufacturer with respect to a forecast. The manufacturer wants to know which wire harness to order from a supplier in order to put in stock.

This is a crucial operation because the ordered harnesses have to fulfill the following requirements:

- For each product variant that is expected to be ordered there must be a matching wire harness in stock
- The wire harnesses in stock must be cost optimal.

This ranges from an individual wire harness for each order to one wire harness for all orders. The computation is based on the following inputs:

- The product portfolio and its properties describing the properties of the product variants and the mutual dependencies of its parts. This is defined by the manufacturer.
- Properties of the wire harnesses describing the dependencies between options in the wire harness. This is defined by the supplier.
- The costs of a wire harness representing the contract with the supplier. The contract defines all involved costs like production cost, stock costs, transportation costs and price deductions.
- An estimation of the expected sales for each product configuration relevant to the wire harnesses.

The current decision process of the manufacturer, which wire harness shall be ordered, is based on a local optimization method. Because of the fact, that the task is a discrete optimization problem, a local optimization can differ significantly from the global optimum (see Chap. 15). This requires a further step for additional optimization.

To compute the global optimum solution, we used the optimization engine of the commercial tool [39]. The tool guarantees that it finds an optimal solution. In order to use this tool we first translated the aforementioned input into a formal model with a defined set of cost attributes. This resulted in the following inputs for the tool, as depicted in Fig. 17.12:

- a model of the products,
- a model of the wire harnesses,
- a model of the costs of the wire harnesses and
- an estimate of the expected sales for each product configuration relevant to the wire harnesses.

Based on the logical representation of the products, the wire harnesses, the costs and the expected sales, the optimization engine computed a set of cost-optimized wire harnesses, i.e. the optimality can be proven mathematically. Because of the

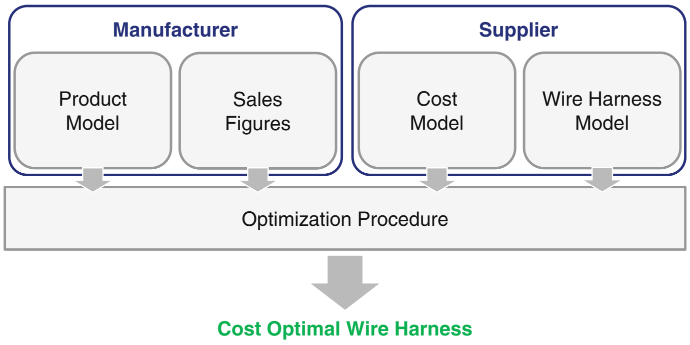


Fig. 17.12 Optimum product model

fact that a discrete optimization is computational expensive, the definition of the formal model is essential, because it has to fulfill two fundamental requirements. First, it must adequately represent the input data. Second, it must be defined in such a way that the optimization procedure works efficiently on the model.

The translation of the input data into a logical model resulted in 300,000 logical formulas. The optimization procedure took about 3 h to compute the global optimal solution. Comparing the optimization results of the tool with the current used method, results in an improvement of at least 15 %.

17.5.3 Car Pool Optimization for Manufacturers and Customers

In the case study car pool optimization, we examined the possibility to compute a cost optimum set of cars for the car pool of a company. The input parameters for this case study were the product model, a cost model for the product model, a given budget and a set of predefined options that every car should have. Then the optimization was on the fuel consumption of the whole car pool.

With current car configurators only one car can be configured with respect to the available options. The configurator computes the price and indicates the fuel consumption of the current configuration. In particular, there is no possibility to select a set of options and the remaining options are set with respect to an optimization procedure.

Further applications are the possibility to define preferences for a configuration and the optimization procedure finds the best car for a given budget. Different customers have their preferences for different kinds of options. For example, safety option, sport options or options causing the least fuel consumption.

Performing optimizations on configurations is a big advantage for both customer and manufacturer. The customer finds the product he wants quickly without bothering about options he is not interested in. The manufacturer on the other hand

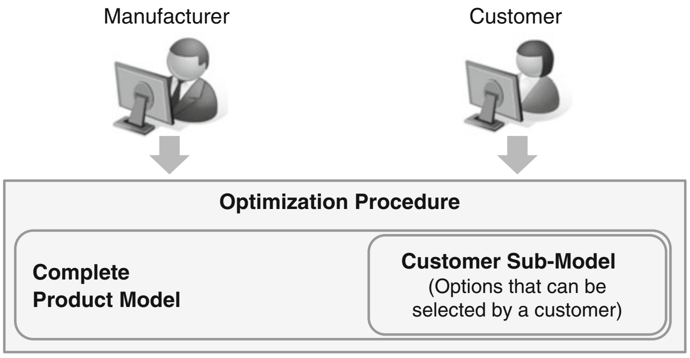


Fig. 17.13 Manufacturer and customer optimize on the same product model

can optimize his product portfolio particularly to the demands of the customers. The optimization enables him to better understand his product portfolio. These insights can be the basis for further decisions, for example launching of marketing offers or invention of new feature options.

Figure 17.13 depicts this case study. The manufacturer and the customer compute optimal configurations based on the same product model. For the customer, a subset of the available configurations is defined. For example, this defines the market offers of the manufacturer. Instead of having different representations of the product portfolio, we show in this case study that manufactures as well as customers can perform their processes on the same product model.

Furthermore, this method provides a marketing driven model based design and development of new products. In this case, the manufacturer starts with a less detailed customer model that defines the customer demands. The model becomes more detailed during the design and development phase of the product. Consequently, the current development model can be optimized with respect to the customer model and verified against the customer model.

17.6 Future Directives

In the late 90s, entering of electronics into products and customer demands for individual products to mass customer market prices caused a rise of the product complexity as indicated in Fig. 17.14. Electronics were shipped with diverse releases and software versions that have to exactly match the hardware parts. This has resulted in a huge number of variants that have to be maintained and have to fulfill specific properties. Further reasons for an explosion of variants are fast changing market demands, legal regulations, and certificates. Nowadays, the complexity of global products drastically rises due to the aforementioned reasons for variety explosion. On the other hand, the general purpose CDCL algorithm

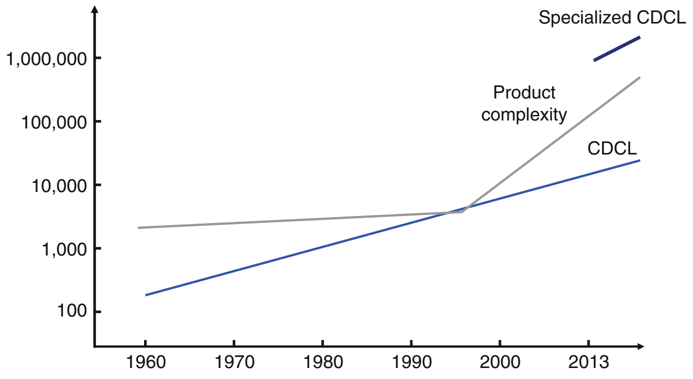


Fig. 17.14 Development of product complexity and the performance of general purpose CDCL procedures and specialized CDCL procedures

(see Sect. 17.3.2.2) has steadily improved such that its implementations could successfully analyze complex products. To be still able to successfully analyze these products with mathematical tools, the CDCL based algorithms can be specialized to the properties of these analyzing problems. An appropriate tool that implements a specialized CDCL algorithm and successfully analyses complex products is available [39].

The case studies in this chapter have shown that algorithms based on the methods depicted in this chapter are able to solve variant problems from various states in the product lifecycle that could not be solved without these. As a consequence, powerful analyzing tools exist that completely and precisely verify complex product models with respect to a diversity of properties involving variants. Furthermore, we expect that models are able to express variant information from other states in the product life cycle and the relations between the states as shown in Fig. 17.9. Examples of other states in the product life cycle are: the requirement and market analysis, design, construction, production, logistics, sales and after-sales.

In the area of model based systems engineering (MBSE) there is a standard modeling language, the systems modeling language (SysML) [42], with a diversity of analyzing tools for several aspects of the systems. In the case of variability management we expect a similar development towards a model based variability management (MBVM). The common variability language (CVL) [14] is an example of the development towards this direction. In the long term this could even go further towards a model based product life cycle management (MB PLM). From such a development we expect a similar impact on the development of products like crash simulations for the development of safe cars.

17.7 Conclusions and Outlook

This chapter has illustrated the crucial importance of variability management. We have presented some of the basic theoretical background and several real-world use cases that have been successfully mastered using methods based on formal logic. Thus, we have shown that the application of a formal logic-based approach to real-world variability problem scenarios is very promising. Although we had to customize the underlying analysis algorithms and find an appropriate way to encode the problem scenario in a formal setting, we are very confident to be able to find such optimizations for most problems of this kind. The more real-world problems we analyze, the more experience we gain in optimizing and customizing our algorithms and problem representations. Although we have to respect and observe the basic research that has been done and that will be done in the area, we are currently at a stage where we need more applied research based on real world examples in real industrial project settings to develop and improve a comprehensive tool suite based on formal logic. This will be one of the key factors enabling manufacturers and product line engineers to develop and master even more complex products and at the same time meet exactly the customer demands. To the best of our knowledge and having analyzed the past, variability reduction certainly is not an option for successful manufacturers or product line engineers in the future in a global competition. The only way for a sustainable variability management and product line engineering is to analyze, optimize, develop and adapt product variability in accordance with the rapidly changing market requirements. Formal approaches as described in this chapter provide the means for achieving this goal.

References

1. Hüttenrauch M, Baum T (2008) Effiziente Vielfalt Die dritte Revolution in der Automobilindustrie. Springer, Berlin
2. Elgh F (2014) Automated engineer-to-order systems. A task oriented approach to enable traceability of design rationale. Int J Agile Syst Manag 7(3–4):324–347
3. McLay A (2014) Re-reengineering the dream: agility as competitive adaptability. Int J Agile Syst Manag 7(2):101–115
4. Alguezaui S, Filieri R (2014) A knowledge-based view of the extending enterprise for enhancing a collaborative innovation advantage. Int J Agile Syst Manag 7(2):116–131
5. Chang D, Chen CH (2014) Understanding the influence of customers on product innovation Int J Agile Syst Manag 7(3–4):348–364
6. Hu S, Ko J, Weyand L, ElMaraghy H, Lien T, Koren Y, Bley H, Chryssolouris G, Nasr N, Shpitalni M (2011) Assembly system design and operations for product variety. CIRP Ann Manuf Technol 60(2):715–733
7. Wallis R, Stjepandić J, Rulhoff S, Stromberger F, Deuse J (2014) Intelligent utilization of digital manufacturing data in modern product emergence processes. In: Cha J et al (eds) Moving integrated product development to service clouds in global economy. Proceedings of the 21st ISPE Inc. international conference on concurrent engineering. IOS Press, Amsterdam, 2014, pp 261–270

8. Kretschmer R, Rulhoff S, Stjepandić J (2014) Design for assembly in series production by using data mining methods. In: Cha J et al. (eds.) Moving integrated product development to service clouds in global economy. Proceedings of the 21st ISPE Inc. international conference on concurrent engineering. IOS Press, Amsterdam, 2014, pp 379–388
9. ElMaraghy H, Schuh G, ElMaraghy W, Piller F, Schönsleben P, Tseng M, Bernard A (2013) Product variety management. CIRP Ann Manuf Technol 62(2):629–652
10. ElMaraghy H, Azab A, Schuh G, Pulz C (2009) Managing variations in products, processes and manufacturing systems. CIRP Ann Manuf Technol 58(1):441–446
11. ElMaraghy W, ElMaraghy H, Tomiyama T, Monostori L (2012) Complexity in engineering design and manufacturing. CIRP Ann Manuf Technol 61(2):793–814
12. Kang KC, Cohen SG, Hess JA, Novak WE, Peterson AS (1990) Feature-oriented domain analysis (foda) feasibility study. Technical report, DTIC Document
13. Pohl K, Böckle G, van der Linden F (2005) Software product line engineering—foundations, principles, and techniques. Springer, Berlin
14. Haugen Ø, Wasowski A, Czarnecki K (2013) Cvl: common variability language. In: Kishi T, Jarzabek S, Gnesi S (eds) Proceedings of the 17th international software product line conference. ACM, New York, p 277
15. Nestor D, O'Malley L, Quigley AJ, Sikora E, Thiel S (2007) Visualisation of variability in software product line engineering. In: Pohl K, Heymans P, Kang KC, Metzger A (eds) 1st International workshop on variability modelling of software-intensive systems (VAMOS 2007). http://www.sse.uni-due.de/vamos/2007/files/vamos07_0038_paper_7.pdf. Accessed 20 Aug 2014
16. Heuer A, Lauenroth K, Müller M, Scheele JN (2010) Towards effective visual modeling of complex software product lines. In: Botterweck G, Jarzabek S, Kishi T, Lee J, Livengood S (eds.) Software product lines: going beyond: Proceedings of the 14th international software product line conference, pp 229–238, SPLC 2010. Jeju Island, Sep 13–17 2010, Springer, Berlin
17. Botterweck G, Thiel S, Nestor D, bin Abid S, Cawley C (2008) Visual tool support for configuring and understanding software product lines. In: Geppert B, Pohl K (eds) Proceedings of the 12th international software product line conference, SPLC 2008. IEEE Computer Society, Los Alamitos, pp 77–86
18. Cawley C, Nestor D, Preußner A, Botterweck G, Thiel S (2008) Interactive visualisation to support product configuration in software product lines. In: Heymans P, Kang KC, Metzger A, Pohl K (eds.) 2nd international workshop on variability modelling of software-intensive systems (VAMOS 2008). http://www.sse.uni-due.de/vamos/2008/papers/VAMOS08_01.pdf. Accessed 20 Aug 2014
19. Schneeweiss D, Botterweck G (2010) Using flow maps to visualize product attributes during feature configuration. In: Botterweck G, Jarzabek S, Kishi T, Lee J, Livengood S (eds) Software product lines: going beyond. Proceedings of the 14th international software product line conference, pp 219–228, SPLC 2010. Jeju Island, 13–17 Sep 2010, Springer, Berlin
20. Czarnecki RK, Helsen S, Eisenecker UW (2004) Staged configuration using feature models. In: Nord RL (ed) Software product lines: 3rd international conference, SPLC 2004, 30 Aug–2 Sep 2004, Boston. vol 3154 LNCS, pp 266–283, Springer, Berlin. http://www.ece.uwaterloo.ca/∼kczarnec/splc04.pdf. Accessed 20 Aug 2014
21. Mendonca M, Wasowski A, Czarnecki K (2009) SAT-based analysis of feature models is easy. In: Muthig D, McGregor JD (eds) 13th international software product line conference, SPLC 2009, 24–28 Aug 2009. San Francisco, ACM international conference proceeding series, 446:231–240, ACM. http://gsd.uwaterloo.ca:8088/SPLOT/articles/mendonca_sat_analysis_splc_2009.pdf. Accessed 20 Aug 2014
22. Antkiewicz M, Czarnecki K (2004) FeaturePlugin: feature modeling plug-in for eclipse. In: Burke G (ed.) Proceedings of the 2004 OOPSLA workshop on eclipse technology eXchange. pp 67–72, ACM. http://gp.uwaterloo.ca/sites/default/files/2004-antkiewicz-feature-modeling-plugin_0.pdf. Accessed 20 Aug 2014

23. Capilla R, Bosch J, Kang KC (2013) Systems and software variability management: concepts, tools and experiences. Springer, Berlin
24. Biere A, Heule M, van Maaren H, Walsh T (2009) Handbook of satisfiability. Frontiers in artificial intelligence and applications, vol. 185. IOS Press, Amsterdam
25. Kang KC, Kim S, Lee J, Kim K, Shin E, Huh M (1998) FORM: a feature-oriented reuse method with domain-specific reference architectures. Ann Softw Eng 5:143–168
26. Benavides D, Segura S, Ruiz-Cortes A (2010) Automated analysis of feature models 20 years later: a literature review. Inf Syst 35(6):615–636. http://www.researchgate.net/publication/223760542_Automated_analysis_of_feature_models_20_years_later_A_literature_review/links/0046352bd57ee8f1c9000000. Accessed 20 Aug 2014
27. Davis M, Putnam H (1960) A computing procedure for quantification theory. J ACM 7 (3):201–215
28. Nelson G, Oppen D (1979) Simplification by cooperating decision procedures. ACM Trans Program Lang Syst 1(2):245–257
29. Ganzinger H, Sofronie-Stokkermans V, Waldmann U (2006) Modular proof systems for partial functions with Evans equality. Inf Comput 204(10):1453–1492
30. Nieuwenhuis R, Oliveras A, Tinelli C (2006) Solving sat and sat modulo theories: From an abstract davis–putnam–logemann–loveland procedure to dpll(t). J ACM 53:937–977, Nov 2006. http://www.divms.uiowa.edu/ftp/tinelli/papers/NieOT-JACM-06.pdf. Accessed 20 Aug 2014
31. Ihlemann C, Jacobs S, Sofronie-Stokkermans V (2008) On local reasoning in verification. In: Ramakrishnan CR, Rehof J (eds) Tools and algorithms for the construction and analysis of systems. 14th international conference, TACAS 2008, held as part of the joint european conferences on theory and practice of software, pp 265–281, ETAPS 2008, Budapest, March 29–April 6 2008. LNCS vol 4963, Springer, Berlin
32. Althaus E, Kruglov E, Weidenbach C (2009) Superposition modulo linear arithmetic sup(la). In: Ghilardi S, Sebastiani R (eds.) Frontiers of combining systems, Proceedings of 7th international symposium, pp 84–99, FroCoS 2009, Trento, September 16–18, 2009. LNCS vol 5749, Springer, Berlin
33. Waldmann U (2001). Superposition and chaining for totally ordered divisible abelian groups (Extended abstract). In: Gore R, Leitsch A, Nipkow T (eds) Automated reasoning : first international joint conference, pp 226–241, IJCAR 2001, LNAI, vol 2083, Siena, 2001. Springer, Berlin
34. Korovin K, Voronkov A (2007) Integrating linear arithmetic into superposition calculus. In: Duparc J, Henzinger TA (eds.) 21st international workshop, CSL 2007, 16th annual conference of the EACSL, Lausanne, Switzerland, 11–15 Sept 2007, vol 4646 LNCS, pp 223–237. Springer, Berlin. http://pdf.aminer.org/000/563/602/a_precedence_based_total_ac_compatible_ordering.pdf. Accessed 20 Aug 2014
35. Weidenbach C, Dimova D, Fietzke A, Kumar R, Suda M, Wischnewski P (2009) SPASS Version 3.5. In: Schmidt R (ed) CADE—22, 22nd international conference on automated deduction, pp 140–145, Montreal, 2–7 Aug 2009, LNCS, vol 5663. Springer, Berlin
36. Suda M, Christoph W, Patrick W (2010) On the saturation of YAGO. IJCAR, pp 441–456
37. Sutcliffe Geoff, Suttner Christian B (2006) The state of CASC. AI Commun 19(1):35–48
38. Larrosa J, Nieuwenhuis R, Oliveras A, Rodriguez-Carbonell E (2009) Branch and bound for boolean optimization and the generation of optimality certificates. In: Kullmann O (ed.) Theory and applications of satisfiability testing—SAT 2009, 12th international conference pp 453–466, SAT 2009, Swansea, 30 Jun–3 Jul 2009, vol 5584 of LNC, Springer, Berlin. http://www.researchgate.net/publication/220944553_Branch_and_Bound_for_Boolean_Optimization_and_the_Generation_of_Optimality_Certificates. Accessed 20 Aug 2014
39. Logic4Business GmbH: www.logic4business.com
40. BigLever Software, Inc.: www.biglever.com
41. Pure systems GmbH. www.pure-systems.com
42. Object Management Group. http://www.omgsysml.org/

43. Bontemps Y, Heymans P, Schobbens PY, Trigaux JC (2004) Semantics of FODA feature diagrams. In: Männistö T, Bosch J (eds.) Proceedings SPLC 2004 workshop on software variability management for product derivation—towards tool support, pp 48–58. http://www.researchgate.net/publication/234080947_Semantics_of_FODA_feature_diagrams/links/00b4952691aadd1ab6000000. Accessed 20 Aug 2014
44. Nieuwenhuis R, Oliveras A, Tinelli C (2004) Abstract DPLL and Abstract DPLL Modulo Theories. LPAR 2004:36–50