Published at Software Quality Journal
Volume 20, Issue 3-4, September 2012, pp 519-565.

The final publication is available at:
http://dx.doi.org/10.1007/s11219-011-9156-5

Quality-aware analysis in product line engineering with the orthogonal variability model

Fabricia Roos-Frantz · David Benavides · Antonio Ruiz-Cortés · André Heuer · Kim Lauenroth

the date of receipt and acceptance should be inserted later

Abstract Software product line engineering (SPLE) is about producing a set of similar products in a certain domain. A variability model documents the variability amongst products in a product line. The specification of variability can be extended with quality information, such as measurable quality attributes (e.g., CPU and memory consumption) and constraints on these attributes (e.g., memory consumption should be in a range of values). However, the wrong use of constraints may cause anomalies in the specification which must be detected (e.g., the model could represent no products). Furthermore, based on such quality information it is possible to carry out quality-aware analyses, i.e., the product line engineer may want to verify whether it is possible to build a product that satisfies a desired quality. The challenge for quality-aware specification and analysis is three-fold. First, there should be a way to specify quality information in variability models. Second, it should be possible to detect anomalies in the variability specification associated with quality information. Third, there should be mechanisms to verify the variability model to extract useful information, such as the possibility to build a product that fulfils certain quality conditions (e.g., is there any product that requires less than 512MB of memory?). In this article, we present an approach for quality-aware analysis in software product lines using the orthogonal variability model (OVM) to represent variability. We propose to map variability represented in the OVM associated with quality information to a constraint satisfaction problem and to use an off-the-shelf constraint programming solver to automatically perform the verification task. To illustrate our approach, we use a product line in the automotive domain which is an example that was created in a national project by a leading car company. We have developed a prototype tool named FaMa-OVM, which works as a proof of concepts. We were able to identify void models, dead and false optional elements, and check whether the product line example satisfies quality conditions.

Fabricia Roos-Frantz · David Benavides · Antonio Ruiz-Cortés
Dept. Computer Languages and Systems, University of Seville,
Avda. Reina Mercedes s/n, 41012, Seville, Spain
E-mail: {fabriciaroos, benavides, aruiz}@us.es

André Heuer · Kim Lauenroth
Paluno - The Ruhr Institute for Software Technology, University of Duisburg-Essen,
Gerlingstr. 16, 45127 Essen, Germany
E-mail: {andre.heuer, kim.lauenroth}@paluno.uni-due.de

Keywords Quality modelling · Software product lines · Quality-aware analysis · Automated analysis · Orthogonal variability model

1 Introduction

Software product line engineering (SPLE) is a paradigm for producing a family of products that share more commonalities than variabilities. This paradigm usually consists of two development processes, namely: *domain engineering* and *application engineering* (Pohl et al, 2005). In domain engineering, the common software artefacts are designed and developed for reuse. In application engineering, the specific products are derived by reusing a set of the aforementioned domain artefacts.

Variability models are central artefacts in all activities of the SPLE. A variability model documents the variability amongst products in the product line, i.e., rules that constrain the possible configurations of artefacts in a product (Chen et al, 2009; Sinnema and Deelstra, 2007; Kang et al, 1990). The configuration of an individual product is done by selecting options in the variability model. Over the past years, several variability modelling techniques have been developed in order to document and manage variability, such as feature modelling, decision modelling and orthogonal variability modelling (Sinnema and Deelstra, 2007; Chen et al, 2009). The *analysis of variability models* deals with the computer-aided extraction of valuable information from variability models (Benavides et al, 2010).

The specification of variability can be extended with measurable quality attributes (e.g., CPU and memory consumption) and constraints on these attributes, in order to express quality information about different products (Benavides et al, 2010). For example, in cases where there are limitations of resources such as memory capacity and CPU time, the derivation of products that does not satisfy those conditions must be avoided. When quality information is added to a variability model a *quality-aware analysis* can be performed. In SPLE, quality-aware analysis is an essential activity to guarantee that the derived software products reach the desired quality. In SPLE, early quality analysis is particularly important, since any anomaly should be identified before the derivation of specific products. The wrong use of constraints may cause anomalies in the specification, leading to contradictory or to misleading information about the scope of the product line. Such anomalies should be detected and avoided to assure that desired products can be configured. If any anomaly is not detected early, all products that were developed based on the anomalous domain artefacts have to be corrected. This can lead to high cost and effort (Pohl et al, 2005).

Currently, there are some approaches that extend feature models with quantitative attributes and constraints (Benavides et al, 2005, 2010). However, quality information associated with the orthogonal variability model (OVM) (Pohl et al, 2005; Metzger et al, 2007) has not been explored in the literature before. The challenge of quality-aware analysis in SPLE using OVM is three-fold. First, a way of expressing quality information has to be provided. Second, possible anomalies in the model should be detected. Third, the variability model should be verified to extract useful information, such as, the possibility to build a product that fulfils certain quality conditions (e.g., is there any product that requires less than 512MB of memory?).

Our main contributions are as follows:

1. To address the first challenge, we present a way to relate the OVM and quality information, which we refer to as OVM+φ. In our approach, quality information consists of quality attributes and constraints on these attributes. Therefore, the approach allows

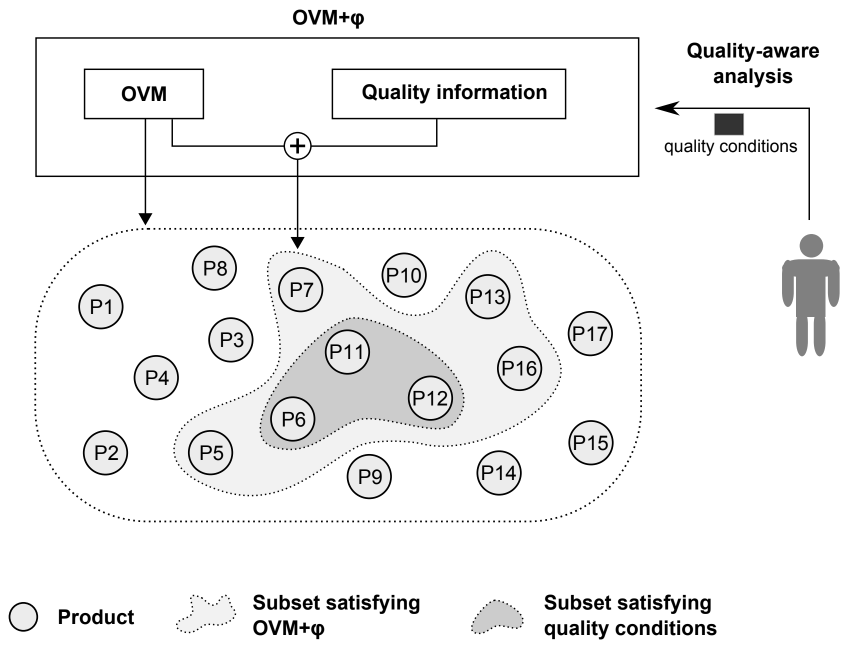


Fig. 1 The approach

modelers to specify levels of quantitative quality attributes for products in SPLE with OVM. These quality attributes are related to the elements that represent variability in the OVM.

2. To address the second challenge, we provide an automated way to detect anomalies in OVM+φ, which is made up of the relationship between OVM and quality information. We support three kinds of anomalies: void model, dead elements, and false optionals. To automate anomaly detection, we present a mapping from OVM+φ to a constraint satisfaction problem. Then, we use an off-the-shelf constraint solver to implement the detection.
3. To address the third challenge, we provide an automated way to carry out analysis operations on SPLE using OVM. We provide analysis operations that can be used to verify quality conditions, ask for an optimal product or the most representative one. A quality condition is any constraint that restricts the value of quality attributes.
4. We have developed a prototype tool named FaMa-OVM, which works as a proof of concepts of our approach. The approach was able to identify *void models*, as well as *dead and false optional* elements for our product line example, which is a non-trivial example. It was also possible to identify whether this product line satisfies quality conditions. Furthermore, we discuss the limitations of our approach.

In this article, we motivate the quality-aware analysis of software product lines by using an example of a Radio Frequency Warner (RFW) product line in the automotive domain. This example was created in a national project by a leading car company. In the example, we use the OVM for variability modelling.

In Figure 1, we show an overview of our approach. An OVM represents a set of possible products. When an OVM is associated with quality information (i.e., quality attributes and constraint on these attributes) which we refer to as OVM+φ, the set of products can be reduced since not all of them satisfy the required quality [1]. Based on this new set of products which takes quality information into account, the engineers of the product line can carry out quality-aware analysis. On the one hand, the OVM+φ specification can be analysed in order to verify possible anomalies. On the other hand, engineers can execute analysis operations to analyse OVM+φ. These operations can use as input a quality condition defined by the engineers, i.e., restrictions on the set of products of OVM+φ. These quality conditions restrict the set of products even more.

The remainder of this article is organized as follows: Section 2 gives an overview of variability modelling techniques, particularly feature models and OVM. Section 3 presents our RFW motivating example. Section 4 discusses how we specify quality information in SPLE using OVM. Section 5 introduces analysis operations that can be performed on OVM+φ, describes the process we use for the automated analysis of OVM+φ, and reports on the mappings from the OVM+φ to a constraint satisfaction problem. Section 6 defines and discusses analysis that extracts information from OVM+φ. Section 7 comments on our tool support, FaMa-OVM, and discusses the obtained analysis results. The related work is discussed in Section 8 and, finally, we discuss limitations and draw our conclusions in Section 9.

2 Background

In SPLE, variability models document the variability of a product line. They provide a set of options that must be selected during derivation of a specific product. Besides, they provide a mechanism to specify rules that constrain the combination of such options. These constraints may come from technical restrictions or any domain decisions. The configuration of products is done by selecting desired and valid options in the variability model during application engineering.

Among the most popular variability modelling techniques is feature modelling, which captures the set of possible products of a product line in a feature model. The first feature model was proposed in 1990 by Kang et al (1990) as part of the Feature-Oriented Domain Analysis (FODA) method. Since then, several extensions of FODA have been proposed.

2.1 Feature Models

Besides documenting variability, feature models also express the commonalities of the product line, i.e., the features that are common to all products. A feature is an increment in program functionality (Batory et al, 2006). Feature models are used to represent product lines by means of a hierarchical decomposition of features, which yields a feature tree. A feature model is composed of two main elements: features and relationships between them, with one of these features being the root. Constraints of the type *requires* and *excludes* between features can be added, leading to additional complexity, thus resulting in a directed acyclic graph.

A common graphical notation is depicted in Figure 2. This feature model example defines a product line, in which every product contains two mandatory features, `A` and `G`. Furthermore, the product line has: (i) one optional feature, `D`, which can be selected or left out at will; (ii) the grouped features `E` and `F` that are possible choices of their parent feature (`D`), in which the *alternative* relationship defines that one and only one of these grouped features

[1] In some cases the number of products could increase, we discuss this in Section 9

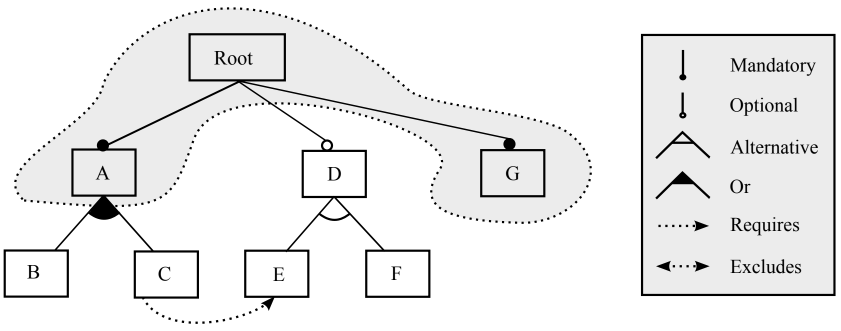


Fig. 2 A feature model example

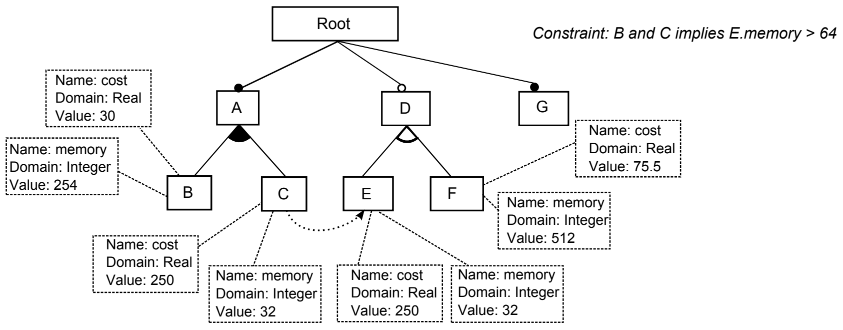

Root
Constraint: B and C implies E.memory > 64
A
D
G
B
C
E
F
Name: cost Domain: Real Value: 30
Name: memory Domain: Integer Value: 254
Name: cost Domain: Real Value: 75.5
Name: memory Domain: Integer Value: 512
Name: cost Domain: Real Value: 250
Name: memory Domain: Integer Value: 32
Name: cost Domain: Real Value: 250
Name: memory Domain: Integer Value: 32

Fig. 3 A sample of a feature model with attributes

can be selected, and (*iii*) the grouped features `B` and `C` that are possible choices of their parent feature (`A`) with the *Or* relationship defining that one or more features of the group must be selected. In addition, the constraints *requires* and *excludes* impose limitations on the possible combinations of features. In this case, the *requires* relationship defines that, when `C` is selected, `E` must be selected as well. The area limited by the grey colour illustrates the features that are common to all products of the product line.

2.2 Extended Feature Models

Some authors have identified the need to extend feature models with extra-functional information such as memory consumption, binary size and development cost (Benavides et al, 2005; Kang et al, 1998; Czarnecki et al, 2005). The purpose of this extension is to add measurable information about the features, which is done by introducing *attributes* to features. By means of these attributes, it is possible to specify quantitative information required to support the feature. As stated by Benavides et al (2010), there is no consensus on a notation to define attributes. However, most proposals agree that an attribute should consist of a name, a domain and a value. Figure 3 shows an example of an extended feature model using the notation proposed by Benavides et al (2005).

This extension enables the inclusion of more complex constraints among features and attributes. For example, it is possible to specify constraints like: *"If feature B and feature C are selected, then memory of feature E must be higher than 64"*.

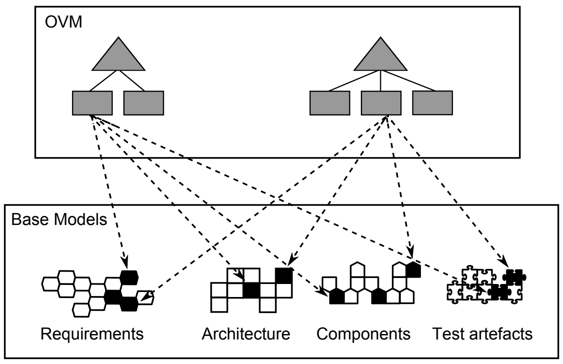
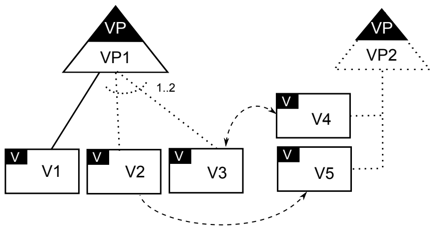
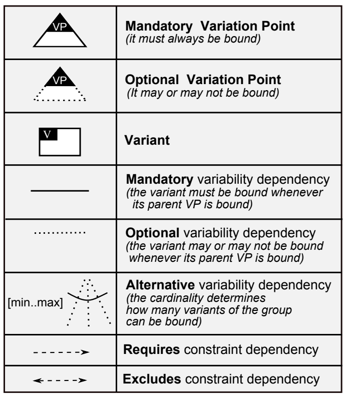


Fig. 4 Orthogonality of OVM (based on (Metzger and Pohl, 2007))





VP	**Mandatory Variation Point** *(it must always be bound)*
VP	**Optional Variation Point** *(It may or may not be bound)*
V	**Variant**
———	**Mandatory** variability dependency *(the variant must be bound whenever its parent VP is bound)*
···········	**Optional** variability dependency *(the variant may or may not be bound whenever its parent VP is bound)*
[min..max]	**Alternative** variability dependency *(the cardinality determines how many variants of the group can be bound)*
- - - - - ->	**Requires** constraint dependency
<- - - - ->	**Excludes** constraint dependency

Fig. 5 OVM notation

2.3 Orthogonal Variability Model

OVM is a modelling language to define the variability of a software product line in an orthogonal way, in other words, it provides a cross-sectional view of the variability across all product line artefacts (Pohl et al, 2005, p.75). OVM interrelates the variability in base models such as requirement models, design models, component models, and test models (see Figure 4). The traceability between OVM and the different types of base models is established through artefact dependencies (dashed lines in Figure 4). In the following, we provide an overview of the OVM. For a complete formal definition of the OVM, we refer the reader to (Metzger et al, 2007).

An OVM is composed of two main elements: *variation points (VP)* and *variants (V)*. In this article, we refer to these OVM elements as variable elements. A variation point documents the aspects that can vary in a product line and are chosen by the customer or engineer of the software product line. A variant is related to a variation point and documents how this variation point can vary. We refer to the variation point and variant relationship as parent-child relationship.

Figure 5 shows an example of an OVM. The variation points have at least one child and each variant has at most one parent. Furthermore, two types of variation points are distinguished, e.g.,`VP1` and `VP2` in Figure 5.

Although feature models are similar to OVMs, they differ mostly in two aspects: in their structure and in the way they relate to quality information. A feature model is only composed of features organised in a single tree and have a single root feature, whilst an OVM is composed of variation points and variants organised in one or more two-level trees. In feature models, the root feature is always mandatory, i.e., it is part of all products and therefore, there is no empty product. On the other hand, in OVMs, variation points can be optional. This allows configuring a product without any variant or variation point. Regarding the second aspect, feature models provide possibility to annotate quality information in the same model, while in OVMs it should be external to the model. We elaborate on this topic in Section 4.

3 Radio Frequency Warner system: motivating example

The RFW product line is used as the motivating example in this article. The aim of systems derived from the RFW product line is to give hints of relevant traffic signs to a driver of a car or truck. The motivation for developing such a product line is the increasing complexity of today's traffic.

The product line is based on the fictional assumption that all traffic signs are equipped with a Radio-Frequency Identification (RFID) tag. This allows the identification of traffic signs when approaching a sign. The transmitted data includes the type of sign (maximum speed, no overtaking, etc.) and the direction of the sign. The functionality of the RFW is realised by a control unit in the car that interacts with other components in the car such as the display and sound system.

An illustration of the functionality of the RFW system can be found in Figure 6: the car is arriving from the east heading to west. Four different signs are in the area: a stop sign, a do-not-enter sign, a no-trucks sign, and a no-parking sign. All these signs are equipped with an active RFID transmitter and each sign knows its direction:

- the stop sign is relevant for all vehicles approaching from the east;
- the do-not-enter sign is relevant for all vehicles approaching from the east and west;
- the no-parking sign is relevant for all vehicles approaching from the west;
- the no-trucks sign is relevant for all trucks approaching from the west and the east.

The information about the direction is encoded in the signal of the traffic signs. If the car, for example, heads to the west, it will receive the signals of all signs. The RFW processes the signals and dismisses the no-parking sign, because it is only relevant for the opposite direction. It signals the stop sign to the driver, since this is the nearest sign to the car that is relevant. During the trip, the RFW system will also show the do-not-enter sign. The no-trucks sign is dismissed for the car. The truck, for example, that is approaching from the west receives the same signals including the no-parking sign in the opposite order, but the RFW does not dismiss the no-trucks sign, because it is relevant for the truck driver.

3.1 System components

An overview of the system can be found in Figure 7. Roughly speaking, the system consists of three components: the RFW display, RF-receiver unit and the RFW control unit. They communicate via a Controller Area Network (CAN) which is a standard interface in the automotive area, standardised by ISO (ISO 11898). However, the CAN is used as a transparent transport gateway. The components can be characterised as follows:

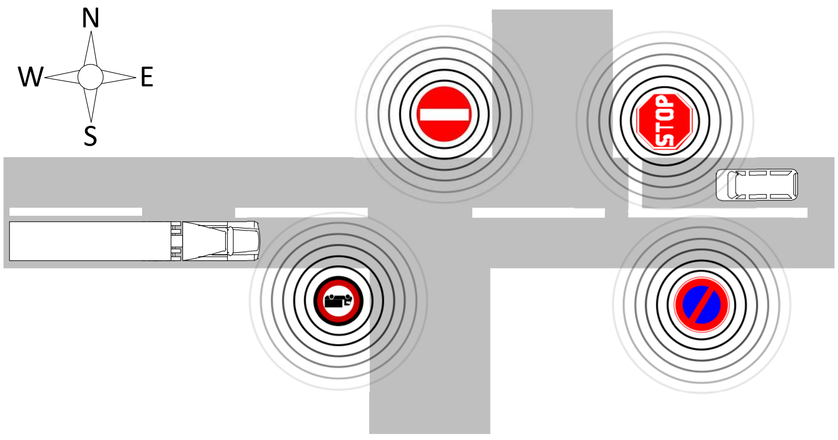


Fig. 6 Functionality of the radio-frequency warner

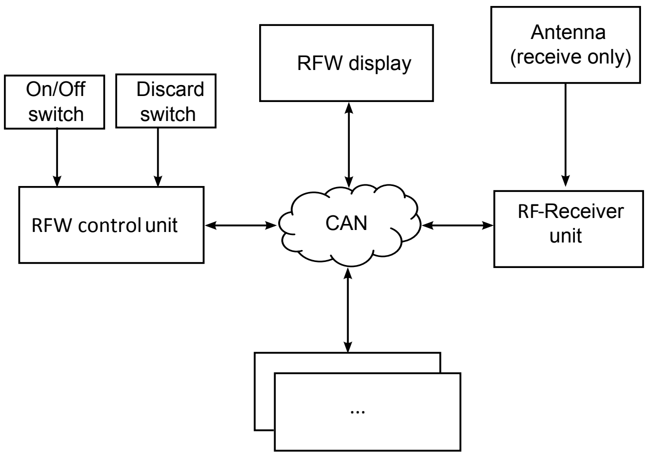


Fig. 7 Radio frequency warner system overview

– *RFW control unit*: the control unit is the main part of the system. It receives the signals of the RF receiver and reacts specifically based on a set of rules.
  – *Discard switch*: the discard switch marks the current signal to be discarded. When the user presses the button, the actual symbol in the display is discarded and the actual warning sound is stopped, and all upcoming signs with the same RFID are dismissed for the next 60 seconds
  – *On/Off switch*: the On/Off switch activates and deactivates the RFW system.
– *RFW display*: the RFW display shows the output of the RFW control unit.
– *RF receiver unit*: the RF receiver unit receives the signals and sends them to the RFW control unit.
  – *Antenna*: the antenna of the RF-receiver unit receives the signals of the active RFID transmitters and decodes them.

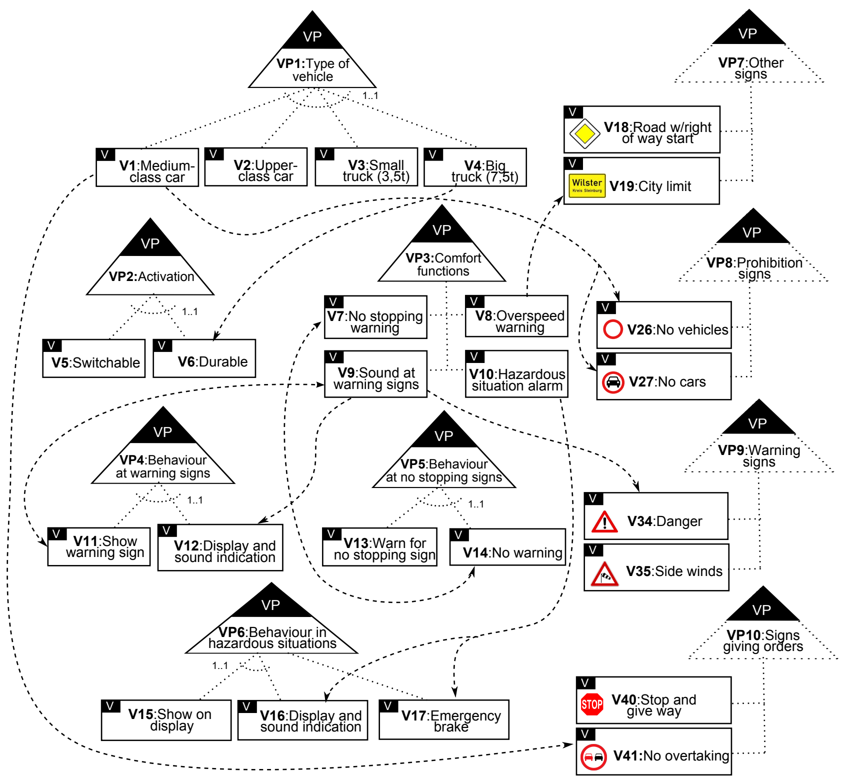


Fig. 8 Excerpt of the RFW orthogonal variability model

3.2 RFW product line

In order to provide a RFW system to customers with different needs, the RFW product line has been created. Figure 8 shows an excerpt of the OVM corresponding to the RFW product line. In this figure, the variation points `VP7:Other signs`, `VP8:Prohibition signs`, `VP9:Warning signs`, and `VP10:Signs giving orders` subsume the different categories of signs that can be detected. For simplification, we show only two signs for each category. The complete OVM diagram with all variation points and variants can be found in Appendix A, Figure 19. Note that in this figure we are omitting several requires and excludes dependencies since it would be too confusing to show all of them. We present the list of these constraints in Appendix A, Table 8.

The main differentiation of the RFW system is made in variation point `VP1:Type of vehicle`. There, one of four different vehicle types has to be chosen. The variation point `VP2:Activation` determines whether the RFW is switchable (i.e., whether it has a switch to turn it on or off) or whether it is turned on instantly and continuously. The variation point `VP3:comfort functions` determines the additional functionality of the RFW. The following comfort functions are available:

– `V7:No stopping warning`: warns the driver if there is an active no stopping sign at the current position and therefore stopping is forbidden.

– `V8:Overspeed warning`: warns the driver if there is a speed limit is in effect and the car is too fast. This requires additional information about the current speed that needs to be received via CAN.
– `V9:Sound at warning sign`: if the car passes a warning sign, the RFW system warns the driver acoustically.
– `V10:Hazardous situation alarm`: warns the driver in a hazardous situation and may take over control, e.g., by initiating an emergency brake. This detection requires much external information, e.g., the actual speed, lateral acceleration, status of the wheels (i.e., blocking, slippage etc.).

The variation point `VP4:Behaviour at warning signs` determines the behaviour, if a relevant warning sign is passed. The system can show the warning sign in the display and it can additionally sound an acoustic warning. The behaviour of the RFW system at a relevant stopping sign is determined by the variation point `VP5:Behaviour at no stopping sign`. The system may warn the driver or not.

The behaviour in a hazardous situation is defined by the variation point `VP6:Behaviour in hazardous situations`. The RFW can show a warning in the display and it can additionally warn the driver with an acoustic signal. Additionally, the system can initiate an emergency brake.

Although the above specification of RFW variability is quite important to guarantee that different customer needs are satisfied by the product line, it does not provide extra-functional information, which is also relevant when developing software products. To satisfy quality conditions regarding, for instance, development cost, memory consumption, or any other quality, this is not enough. Extra-functional information must be specified and related to the RFW variability. Based on this specification, a quality-aware verification can be performed to ensure that all products fulfill the stakeholders needs. In order to specify quality information for the RFW product line, to relate it to the variability, and to verify that its products satisfy the quality conditions, some challenges arise. These are described in the next section.

4 Expressing quality information

In the RFW product line, we have identified many variation points regarding functionalities of the product line systems, but no quality information. Quality information such as the specification of the development cost of the comfort function or the power of the sensor required by different signals, cannot be expressed directly in the OVM.

In feature models, the attributes annotate features with quality information (see Figure 3). In the OVM, this is different, since OVM documents the variability of base models. There are two different possibilities when relating quality information with the OVM which represent two different problems: (i) the OVM is directly related to a quality model including quality information, considering this model as a base model in the OVM terminology, see Figure 9 (a); (ii) the quality information refers to a base model (e.g., architecture, requirements or configuration models) so the OVM is not directly related to a quality model, see Figure 9 (b). In this article we address the second problem.

In the following, we consider the case that the base model would be a configuration model and this model is related to quality information. Roughly speaking, a configuration model is made up of a set of components (e.g., features or variants) and rules that constrain the possible combinations of these components. We assume that rules can be represented in an OVM, and that the relationship between elements in both models (OVM and configuration model) is one-to-one (see discussion in Section 9). The specific configuration model

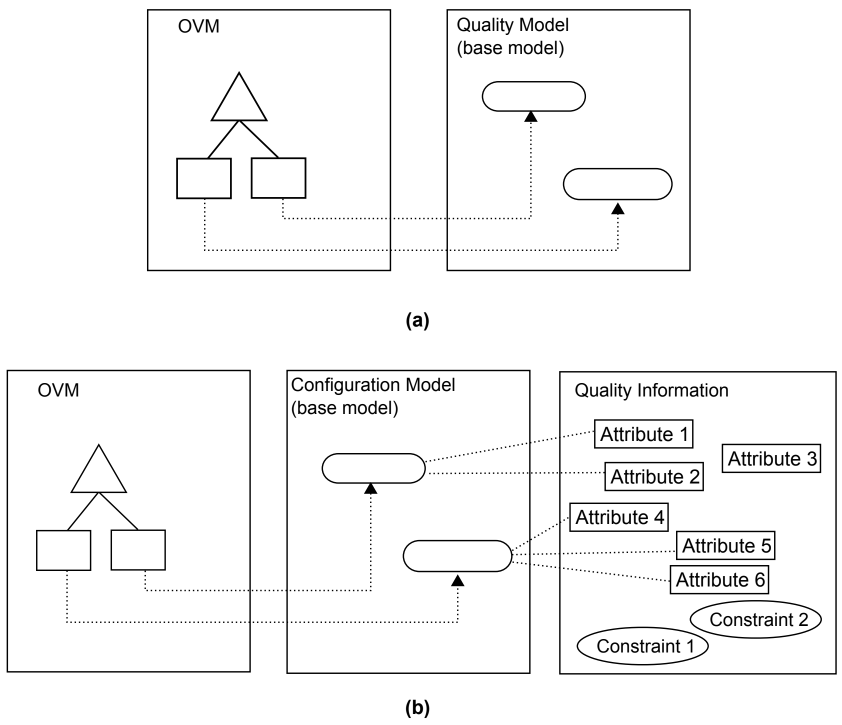


Fig. 9 (a) OVM documenting variability of a quality model; (b) OVM documenting variability of a configuration model that has quality information

depends on the language used to model the configuration problem (Finkel and O'Sullivan, 2011; Felfernig et al, 2000). As our approach is independent of the configuration model used and for the sake of simplicity, we omit the configuration model from the subsequent figures and text. We relate the OVM directly to quality information, hereafter called OVM+φ (see Figure 10), still considering the second problem described in the previous paragraph, i.e., we are still not documenting the variability of a quality model.

Although it is out of the scope of this article to provide a rigorous syntax and semantics of a language to define quality information in OVMs (this would require a parallel research), we assume that such information can be mapped into a constraint satisfaction problem (CSP), thus enabling the automation of our approach. A CSP is defined as a set of variables, a set of domains for those variables, and a set of constraints restricting the values of those variables (Tsang, 1993). For example, suppose $x1, x2, x3$ are variables of a CSP, all with domains in $[1,2,3]$, and $(x1 = x2), (x2 < x3)$ being the constraints. A solution to this CSP is an assignment to every variable of some value in its domain such that it does not violate any of the constraints. Therefore, a possible solution to this CSP is ((x1 = 1), (x2 = 1), (x3 = 2)).

In Figure 11, we provide a high-level conceptual model for describing the main elements of a quality information language such as the one we use in our approach. A quality information language is represented by a set of attributes with their respective domains, and/or a set of constraints on these attributes. In the following, we describe these elements and how they are related to the OVM.

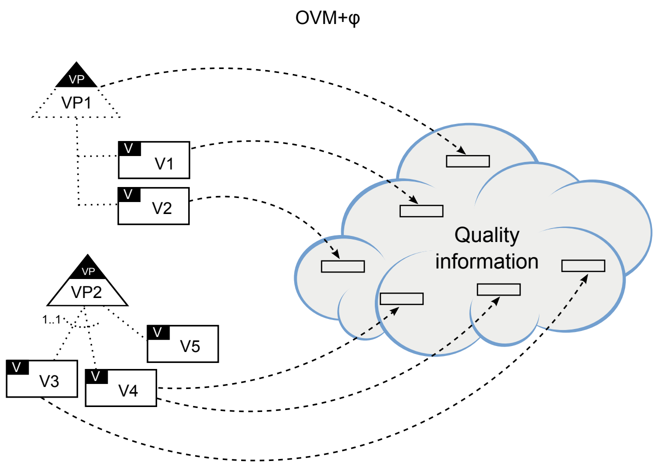


Fig. 10 OVM+φ: relationship between OVM and quality information

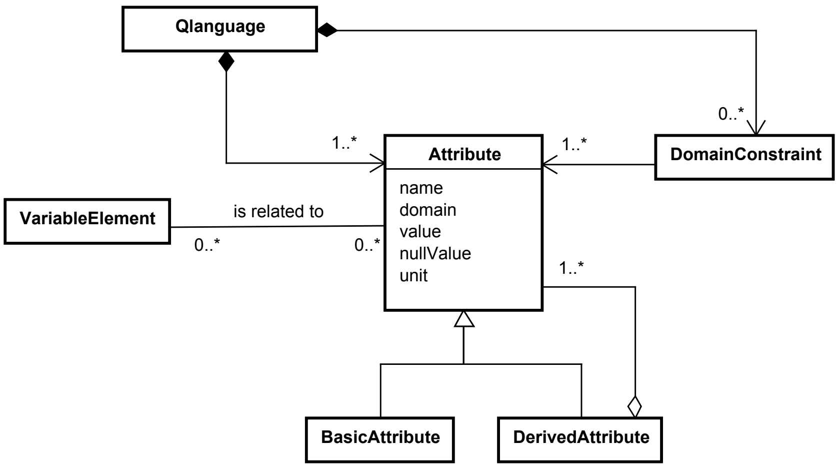


Fig. 11 Conceptual model for describing quality information

4.1 Quality attributes

In our approach, we define a quality attribute as a measurable property of an artefact. We consider only those properties that can be quantified and technically defined. For example, the memory consumption or the accuracy of an antenna. As stated by Benavides et al (2010), most proposals agree that an attribute should consist of a name, a domain and a value. We have relied on this statement to specify the attributes used in our approach. An attribute has a name, a domain, a value, a nullValue, and unit. name denotes the name of the attribute which does not need to be unique since different artefacts can have different attributes with the same name. domain denotes the range of values that the attribute may hold such as Reals, Integers, and any range (e.g., [1..512]); value denotes the attribute value which will depend on the concrete type of attribute (we elaborate more on it later). nullValue denotes the value that must be taken by the attribute when the variant with which the attribute is related is

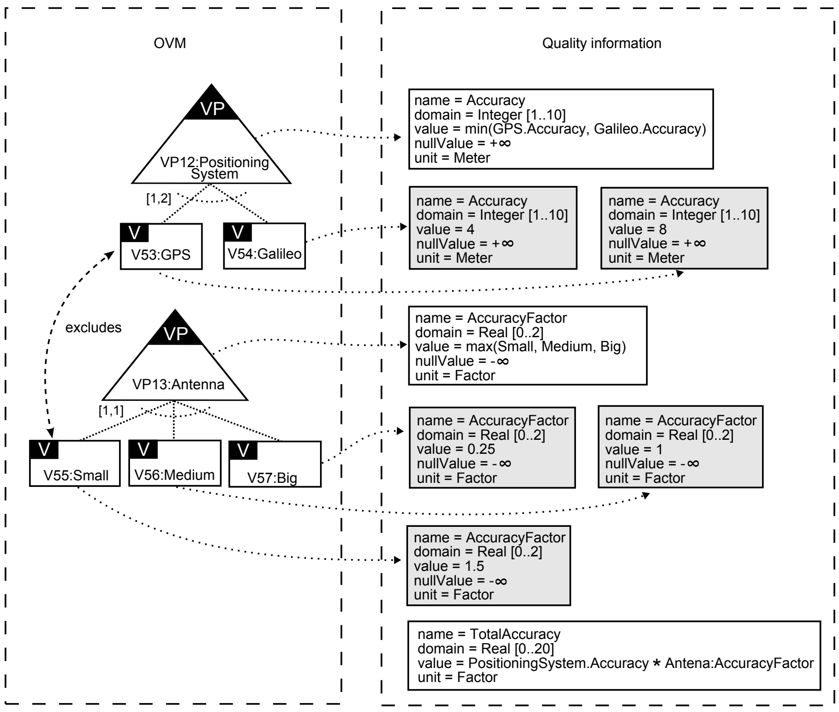


Fig. 12 Example of basic, derived, and global attribute

not selected. unit denotes a determinate quantity such as meters, seconds, currency, and kilobytes, adopted as a standard for measurement.

We distinguish two kinds of attributes depending on how their values are calculated:

– *Basic attribute*. The value of a basic attribute is a *base measure* (Garcia et al, 2006), i.e., a measure that does not depend upon any other measure.
– *Derived attribute*. The value of a derived attribute is determined by a function over other attribute values.

An attribute can be related to zero or more variable element in the OVM. In the same manner, a variable element can be related to zero or more attributes. We refer to an attribute as a *global attribute* when it is not related to a variable element in the OVM; it is a composition of any other attributes. We refer to a relationship between a variable element and an attribute as variable-element.attribute, where variable-element denotes the name of the variable element which must be unique and attribute denotes the name of the attribute. For example, `V53:GPS.Accuracy` defines the relationship between `V53:GPS` and `Accuracy`.

To illustrate quality attributes, we use the example in Figure 12. In this example, we can see that the RFW product line offers two different types of positioning systems: `V53:GPS` and/or `V54:Galileo`. The positioning systems have different accuracies, thus we define a basic attribute to express their accuracy. The GPS system has an accuracy of 8 meters, while Galileo has 4 meters of accuracy.

In Figure 12, there are two derived attributes: `Accuracy` and `AccuracyFactor`, related to `VP12:PositioningSystem`, and `VP13:Antenna`, respectively. The former expresses the system accuracy regarding the type of positioning system selected, and it is the minimum value between `V53:GPS.Accuracy` and `V54:Galileo.Accuracy`; the latter expresses the accuracy factor of the selected antenna which is obtained by the maximum value between `Small.AccuracyFactor`, `Medium.AccuracyFactor` and `Big.AccuracyFactor`. Finally, the `TotalAccuracy` is a global attribute because it is not connected to any variable element. The global attribute represents a quality property of the product line as a whole. Thus, in the case of `TotalAccuracy`, it represents the resulting accuracy of the system, which is the product of the selected positioning system accuracy and the selected quantifier. With this specification, it would, for example, be possible to get the same overall accuracy with a bigger antenna and GPS and a medium size antenna and Galileo.

The function used to calculate the values of derived attributes depends on the solver used to automate the approach; furthermore, it is domain dependent. The resulting value of a function depends on whether the variable element related to it is selected or not. Therefore, when an attribute is involved in a function its `nullValue` must be neutral to such a function. Each function must be handled specifically and suitable neutral values must be defined. Let us observe, for example, the function defined in the `VP12:PositioningSystem.Accuracy` attribute. In the case where both positioning systems, `V54:Galileo` and `V53:GPS`, are selected, the function will return the minimum value between their accuracy, resulting in value 4. However, if one of the variants is not selected, for example `V53:GPS`, the value of `V53:GPS.Accuracy` must have a neutral value with regard to the *min* function. In this case, we can use the $+\infty$ as a neutral value because it is bigger than any real number. In the RFW example, we use aggregate functions such as *sum*, *min* and *max* and also functions with the operators $+$ and $*$.

The list of attributes identified for the RFW product line and their descriptions can be found in Table 1. Their values are not shown in this table because they depend on the association with the variable elements in the OVM. The values are shown in Table 2, Table 3, and Table 4; they were defined by the engineers of the RFW product line. Table 2 shows the values taken by the basic attributes. The variants are listed in the first column of the table, and the attribute names are listed along the first row. The cells indicate the values taken by each attribute when related to the corresponding variant. They are shown in the form *value|neutral-value*. Cells marked with "–" indicate that the attribute is not related to the variant.

In the RFW example, all derived attributes are related to variation points. Their values are shown in Table 3. The variation points are listed in the first column, and the attribute names are listed along the first row. The cells indicate which function is used to calculate each attribute value when related to the corresponding variation point. Those cells marked with "–" indicate that the attribute is not related to the variation point. Values are shown in the form *value|neutral-value*. As these attributes are involved in the values of the global attributes (see Table 4), their null values are defined as $+\infty$, $-\infty$ or 0. In the case that the variation point is selected, the attributes can take as values the functions *min*, *max* or *sum*. They are defined as follows:

- $min(v_1.attribute, ..., v_n.attribute),\ where\ \{v_1, ..., v_n\} \subseteq childrenOf(vp)$
- $max(v_1.attribute, ..., v_n.attribute),\ where\ \{v_1, ..., v_n\} \subseteq childrenOf(vp)$
- $\sum_{i=1}^{n} v_i.attribute,\ where\ \{v_1, ..., v_n\} \subseteq childrenOf(vp)$

Consider that $childrenOf(vp)$, with vp belonging to the set of variation points returns the set of children of vp, and $n \leq |childrenOf(vp)|$. Next, we provide some examples.

```
VP12:PositioningSystem.Accuracy = min(GPS.Accuracy,
```

Table 1 Quality attributes in the RFW product line

Name	Description	Domain	Unit
1. Accuracy	Specifies the accuracy of the positioning system to locate the position of the car	Integer [1..10]	meters
2. AccuracyFactor	Specifies a quantifier for the antenna	Real [0..2]	meters
3. TotalAccuracy	Specifies the accuracy offered by the system. It is calculated by relating the accuracy and the accuracyFactor attributes	Real[0..20]	meters
4. Memory	Specifies the memory size of the control unit that is needed to process the traffic sign	Integer [1..512]	kilobytes
5. TotalMemory	Specifies the total of memory required by a system. It is calculated by aggregating the memory attributes	Integer[1..512]	kilobytes
6. ROM	Specifies the ROM size of the control unit utilised by a specific variant	Integer [1..512]	kilobytes
7. TotalROM	Specifies the ROM size of the control unit. The more traffic signs are recognisable, the bigger the ROM size has to be to save the different types of signs and the required action for the traffic sign. It is calculated by aggregating the ROM attributes	Integer [1..512]	kilobytes
8. Range	It concerns the power of a sensor and specifies the distance from which the sensor is capable to detect a traffic sign. The higher the value is, the earlier a traffic sign can be detected	Integer	meters
9. Latency	Specifies the latency required by a variant. Latency means the elapsed time between the firing of an event and the feedback given to the user.	Integer [200..800]	milliseconds
10. TotalLatency	Specifies the latency that has to be guaranteed by the system. It is calculated by aggregating the latency attributes	Integer [200..800]	milliseconds
11. Cost	Cost of the specific variant	Real [1..500]	monetary unit
12. TotalCost	Specifies the total cost of a system. It is calculated by aggregating the cost attributes	Real [1..500]	monetary unit
13. Cycle	Specifies the maximum recognition time required by a variant.	Integer[10..500]	milliseconds
14. RecognitionTime	Specifies the maximum recognition time that the system has to ensure. It is calculated by aggregating the cycle attributes	Integer [10..500]	milliseconds

```
                                    Galileo.Accuracy)

VP13:Antenna.AccuracyFactor = max(Small.AccuracyFactor,
                                  Medium.AccuracyFactor,
                                  Big.AccuracyFactor)

VP2:Activation.Memory = V5:Switchable.Memory + V6:Continuously.Memory
```

The equations for the values of global attributes are shown in Table 4. Except for the `TotalAccuracy`, the other global attributes are calculated using an aggregate function in the set of variation points. In the following we elaborate on their meaning:

– TotalAccuracy: represents the overall accuracy of a given product which is computed by multiplying the accuracy of the positioning system by the antenna accuracy factor. For

Table 2 Values of basic attributes when associated with variants

	Accuracy	Accuracy Factor	Memory	ROM	Range	Latency	Cost	Cycle
V5:Switchable	–	–	2 | 0	2 | 0	–	–	2 | 0	–
V6:Continously	–	–	2 | 0	2 | 0	–	–	0.2 | 0	–
V7:No stopping warning	–	–	–	8 | 0	–	–	0.5 | 0	–
V8:Overspeed warning	–	–	–	16 | 0	–	–	0.5 | 0	–
V9:Sound at warning signs	–	–	–	4 | 0	–	–	1 | 0	–
V10:Hazardous situation alarm	–	–	–	16 | 0	–	–	1 | 0	–
V11:Show warning sign	–	–	2 | 0	4 | 0	–	400 | $+\infty$	1 | 0	–
V12:Display and sound indication	–	–	2 | 0	4 | 0	–	400 | $+\infty$	1 | 0	–
V13:Warn for no stopping sign	–	–	2 | 0	4 | 0	–	500 | $+\infty$	0.5| 0	–
V14:No warning	–	–	2 | 0	0 | 0	–	500 | $+\infty$	0.2 | 0	–
V15:Show on display	–	–	8 | 0	8 | 0	–	350 | $+\infty$	1 | 0	–
V16:Display and sound indication	–	–	8 | 0	8 | 0	–	350 | $+\infty$	1 | 0	–
V17:Emergency brake	–	–	16 | 0	32 | 0	–	350 | $+\infty$	3 | 0	–
V18:Road w/ right of way start	–	–	4 | 0	4 | 0	–	–	0.2 | 0	100 | $+\infty$
V19:City limit	–	–	2 | 0	4 | 0	–	–	0.2 | 0	50 | $+\infty$
V20:Crossroads	–	–	2 | 0	4 | 0	–	–	0.2 | 0	100 | $+\infty$
V21:Home zone entry	–	–	4 | 0	4 | 0	–	–	0.2 | 0	100 | $+\infty$
V22:Road w/ right of way end	–	–	4 | 0	4 | 0	–	–	0.2 | 0	100 | $+\infty$
V23:End of city limit	–	–	4 | 0	4 | 0	–	–	0.2 | 0	50 | $+\infty$
V24:Traffic has priority	–	–	2 | 0	4 | 0	–	–	0.2 | 0	75 | $+\infty$
V25:Home zone end	–	–	4 | 0	4 | 0	–	–	0.2 | 0	100 | $+\infty$
V26:No vehicles	–	–	2 | 0	4 | 0	–	–	0.2 | 0	50 | $+\infty$
V27:No cars	–	–	2 | 0	4 | 0	–	–	0.2 | 0	100 | $+\infty$
V28:No vehicles over max width $>$ Xm	–	–	8 | 0	8 | 0	–	–	0.2 | 0	200 | $+\infty$
V29:No vehicles w/ weight $>$ 3.5t	–	–	2 | 0	4 | 0	–	–	0.2 | 0	100 | $+\infty$
V30:No vehicles over max gross weight g $>$ Xt	–	–	8 | 0	8 | 0	–	–	0.2 | 0	200 | $+\infty$
V31:Do not enter	–	–	2 | 0	4 | 0	–	–	0.2 | 0	100 | $+\infty$
V32:No vehicles over max height h $>$ Xm	–	–	8 | 0	8 | 0	–	–	0.2 | 0	200 | $+\infty$
V33:No stopping	–	–	2 | 0	4 | 0	–	–	0.2 | 0	100 | $+\infty$
V34:Danger	–	–	2 | 0	4 | 0	–	–	0.5 | 0	100 | $+\infty$
V35:Side winds	–	–	2 | 0	4 | 0	–	–	0.5 | 0	100 | $+\infty$
V36:Slippery road	–	–	2 | 0	4 | 0	–	–	0.5 | 0	100 | $+\infty$
V37:Risk of ice	–	–	2 | 0	4 | 0	–	–	0.5 | 0	100 | $+\infty$
V38:Bend	–	–	2 | 0	4 | 0	–	–	0.5 | 0	100 | $+\infty$
V39:Traffic queues	–	–	2 | 0	4 | 0	–	–	0.5 | 0	100 | $+\infty$
V40:Stop and give way	–	–	2 | 0	4 | 0	–	–	0.2 | 0	50 | $+\infty$
V41:No overtaking	–	–	4 | 0	4 | 0	–	–	0.2 | 0	50 | $+\infty$
V42:No overtaking end	–	–	4 | 0	4 | 0	–	–	0.2 | 0	50 | $+\infty$
V43:No overtaking vehicles $>$ 3.5t	–	–	4 | 0	4 | 0	–	–	0.2 | 0	50 | $+\infty$
V44:End of prohibitions	–	–	4 | 0	4 | 0	–	–	0.2 | 0	50 | $+\infty$
V45:Yield	–	–	2 | 0	4 | 0	–	–	0.2 | 0	50 | $+\infty$
V46:Maximum speed X Km/h	–	–	8 | 0	8 | 0	–	–	0.2 | 0	200 | $+\infty$
V47:One way	–	–	2 | 0	4 | 0	–	–	0.2 | 0	50 | $+\infty$
V48:Maximum speed of X Km/h end	–	–	8 | 0	8 | 0	–	–	0.2 | 0	200 | $+\infty$
V49:No overtaking vehicles $>$3.5t end	–	–	4 | 0	4 | 0	–	–	0.2 | 0	50 | $+\infty$
V50:Low	–	–	–	–	20 | $-\infty$	–	10 | 0	–
V51:Medium	–	–	–	–	45 | $-\infty$	–	35 | 0	–
V52:High	–	–	–	–	70 | $-\infty$	–	50 | 0	–
V53:GPS	8 | $+\infty$	–	–	–	–	–	–	–
V54:Galileo	4 | $+\infty$	–	–	–	–	–	–	–
V55:Small	–	1.5 | $-\infty$	–	–	–	–	15 | 0	–
V56:Medium	–	1 | $-\infty$	–	–	–	–	0.5| 0	–
V57:Big	–	0.25 | $-\infty$	–	–	–	–	50 | 0	–

Table 3 Values of derived attributes when associated with variation points

	Accuracy	Accuracy Factor	Memory	ROM	Range	Latency	Cost	Cycle
VP2:Activation	–	–	sum $\|0$	sum $\|0$	–	–	sum $\|0$	–
VP3:Comfort functions	–	–	–	sum $\|0$	–	–	sum $\|0$	–
VP4:Behaviour at warning signs	–	–	sum $\|0$	sum $\|0$	–	min $\|+\infty$	sum $\|0$	–
VP5:Behaviour at no stopping signs	–	–	sum $\|0$	sum $\|0$	–	min $\|+\infty$	sum $\|0$	–
VP6:Behaviour in hazardous situations	–	–	sum $\|0$	sum $\|0$	–	min $\|+\infty$	sum $\|0$	–
VP7:Other signs	–	–	sum $\|0$	sum $\|0$	–	–	sum $\|0$	min $\|+\infty$
VP8:Prohibition signs	–	–	sum $\|0$	sum $\|0$	–	–	sum $\|0$	min $\|+\infty$
VP9:Warning signs	–	–	sum $\|0$	sum $\|0$	–	–	sum $\|0$	min $\|+\infty$
VP10:Signs given orders	–	–	sum $\|0$	sum $\|0$	–	–	sum $\|0$	min $\|+\infty$
VP11:Sensor power	–	–	–	–	max $\|-\infty$	–	sum $\|0$	–
VP12:Positioning system	min $\|+\infty$	–	–	–	–	–	–	–
VP13:Antenna	–	max $\|-\infty$	–	–	–	–	sum $\|0$	–

Table 4 Equations for the values of global attributes

Name	**Value**
TotalAccuracy	$PositioningSystem.Accuracy * Antenna.AccuracyFactor$
TotalMemory	$\sum_{j=1}^{k} vp_j.Memory$, where $vp_j.Memory \in$ OVM+φ
TotalROM	$\sum_{j=1}^{k} vp_j.ROM$, where $vp_j.ROM \in$ OVM+φ
TotalLatency	$min(vp_1.Latency, ..., vp_k.Latency)$, where $vp_1.Latency, ..., vp_k.Latency \in$ OVM+φ
TotalCost	$\sum_{j=1}^{k} vp_j.Cost$, where $vp_j.Cost \in$ OVM+φ
RecognitionTime	$min(vp_1.Cycle, ..., vp_k.Cycle)$, where $vp_1.Cycle, ..., vp_k.Cycle \in$ OVM+φ

$k \leq$ number of variation points $\in$ OVM+φ

example, a product of the RFW product line that has `GPS` and a `medium` antenna offers an overall accuracy of 8, since V53:GPS.Accuracy = 8 and V56:Medium = 1.

– TotalMemory: represents the total memory required by a given product, which is computed by the sum of all attributes `Memory` related to variation points.
– TotalROM: represents the total ROM required by a given product, which is computed by the sum of all attributes `ROM` related to variation points.
– TotalLatency: represents the maximum time that a given product takes to provide feedback, which corresponds to the minimum value amongst the attributes `Latency` related to variation points. For example, the minimum value between VP1.Latency = 350 and VP2.Latency = 500 is 350. Then, the maximum time this product should take to provide feedback to the user is 350 milliseconds.
– TotalCost: represents the cost of a given product, which is computed by the sum of all attributes `Cost` related to variation points.
– RecognitionTime: represents the maximum time that a given product takes to recognize a signal, which corresponds to the minimum value amongst the attributes `Cycle` related to variation points. For example, the minimum value between VP1.Cycle = 50 and VP2.Cycle = 100 is 50. Then, the maximum time this product should take to recognize a signal is 50 milliseconds.

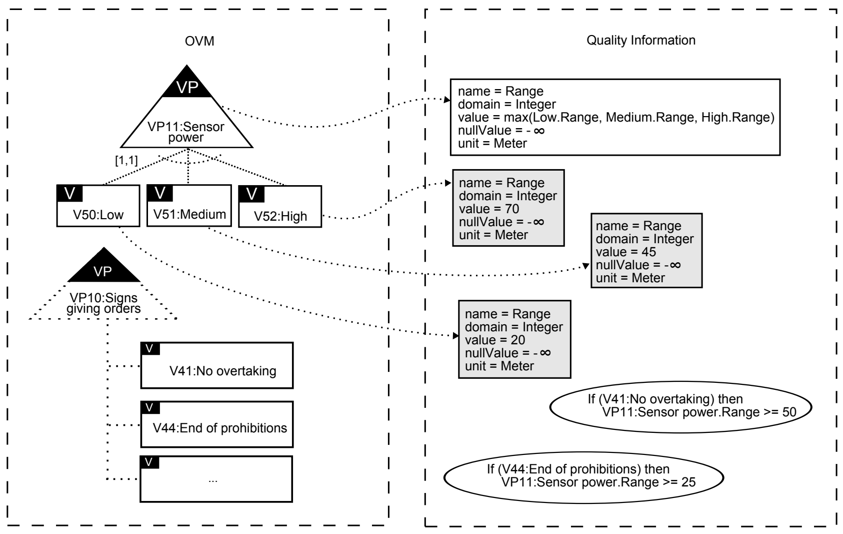


Fig. 13 An example with constraints on attributes

4.2 Domain constraints

Domain constraints are constraints on attributes that limit the possible configuration of products. These constraints may come from resource limitations (e.g., the maximum memory consumption allowed) or any domain relevant restriction, e.g., all products derived from the RFW product line must provide feedback to the user before passing the signals, otherwise the system is useless. Therefore, domain constraints can be defined on quality attributes to avoid building unsuitable products.

In the case of the RFW product line, for example, the traffic signs must be detected by the sensor from a given distance before passing the traffic sign. This distance must be reasonable to allow the product to provide feedback to the user within an expected time. Therefore, some constraints on the attribute `VP11:Sensor power.range` must be defined. To guarantee that a RFW product can successfully detect the `V44:End of prohibitions` sign, the sensor has to have a range of at least 25 meters. However, to detect the `V41:No overtaking` sign, the sensor must be able to detect the signal at least 50 meters before passing the sign. Thus, the constraints on attributes represented within ellipses in Figure 13 are specified to prevent the configuration of unsatisfactory products.

The syntax of domain constraints depends on the solver used to automate the analysis. Domain constraints are predicates over attributes that can be evaluated to true or false depending on the attribute values. The neutral values for attributes must also be considered in constraint definitions. In the RFW product line, we have identified some required domain constraints. A complete list of these constraints can be found in Appendix A, Figure 20.

5 Detecting anomalies in OVM+φ by means of CSP

A typical problem that comes to light when specifying OVM is the wrong use of constraints. A wrong constraint modelling can cause anomalies in the specification, which are difficult to detect in practice. In addition to requires and excludes constraints, domain constraints

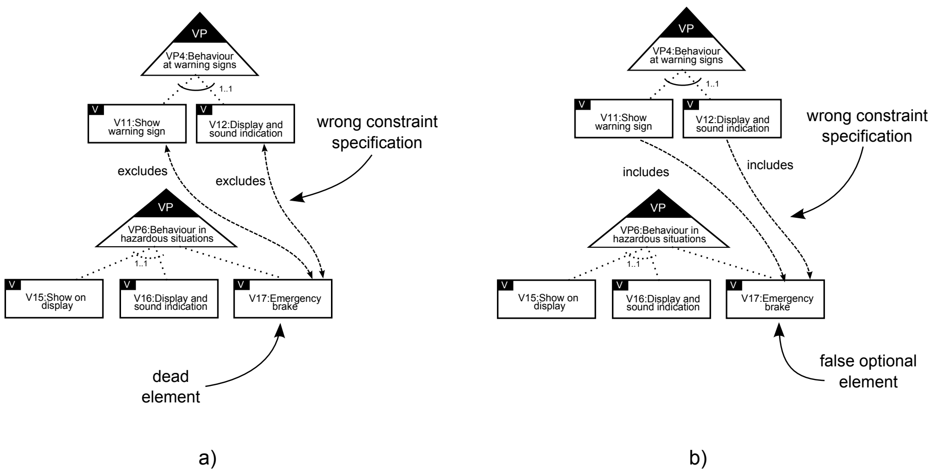

VP
VP4:Behaviour at warning signs
1..1
V
V11:Show warning sign
V
V12:Display and sound indication
wrong constraint specification
excludes
excludes
VP
VP6:Behaviour in hazardous situations
1..1
V15:Show on display
V16:Display and sound indication
V17:Emergency brake
dead element
a)
includes
includes
false optional element
b)

Fig. 14 Anomalies in OVM: a) Dead element, b) False optional element

can also cause anomalies. Therefore, before analysing quality conditions, we have to detect possible anomalies in the OVM+φ. For example, anomalies such as "*the model does not allow the derivation of any product that respects all the specified dependencies amongst variants*", should be detected.

The RFW product line has 13 variations points, 4 out of which are optional, and 57 variants. Consequently, there are 57 variability dependencies between variation points and variants, 65% of them are optional and 35% are alternative. Furthermore, the product line has 34 requires and 4 excludes relationships between variable elements, 225 attributes (18 out of them are derived), and 72 quality domain constraints. Consequently, in order to manually detect anomalies in this model is a tedious and error-prone task. This task is even more complicated if we consider that the products should respect domain constraints.

There are three kinds of anomalies we intend to check in the specification of OVM+φ; namely *void model*, *dead variable element*, and *false optional*. These anomalies were already identified and supported by other approaches when using feature models (Benavides et al, 2010; Trinidad et al, 2008a). In this article we propagate those results to the OVM context.

– *Void model.* A model is void when it is not possible to derive any valid product, i.e., a product that respects the rules specified in the OVM+φ.
– *Dead variable element.* Using constraints wrongly can generate a dead variable element, i.e., it never appears in any valid product. Figure 14. a shows a false constraint between variant `V12:Display and sound indication` and variant `V17:Emergency brake`. The `V17:Emergency brake` behaviour will not be part of a product regardless of whether `V11:Show warning signs` or `V12:Display and sound indication` is selected. This situation gives a false view of the product line scope, since that the optional dependency between `VP6:Behaviour at hazardous situation` and `V17:Emergency brake` determines that it should be possible to configure products with and without `V17:Emergency brake`. When the false constraint on this `V17:Emergency brake` is specified, this functionality will never appear in a product of the RFW product line, and thus will lead to a dead variant.
– *False optional.* Verifies whether a variable element is false optional or not. A variable element is false optional if it is modelled as optional, yet appears in all valid products.

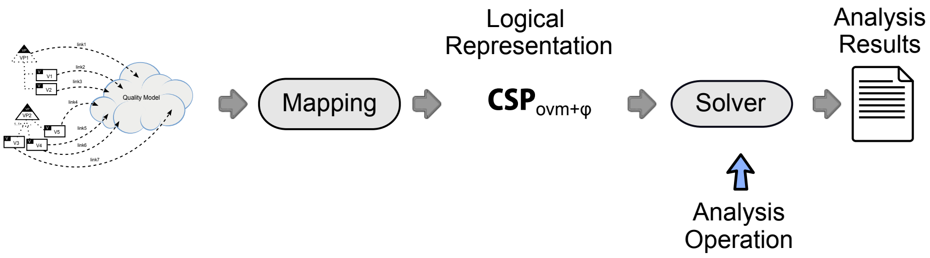


Fig. 15 Process for the automated analysis of OVM+φ using CSP

This anomaly also gives a false view of the product line scope. Figure 14.b shows how the variant `V17:Emergency brake` can become a false optional.

The purpose of automated analysis of feature models is extracting information from feature models using automated mechanisms (Batory et al, 2006). Benavides et al (2010) define a conceptual framework, in which they propose a process for the automated analysis of feature models. Based on this framework, we define the process presented in Figure 15 as the whole process to automate the analysis of OVM+φ. The process starts by mapping the OVM+φ to a CSP, which is the logical representation we use to automate the analysis (hereinafter, this mapping is referred to as $\psi_{ovm+\varphi}$). Afterwards, we define analysis operations, which observe the properties of the $\psi_{ovm+\varphi}$ model without modifying it; they take a $\psi_{ovm+\varphi}$ model and/or a configuration as input and provide a response. An off-the-shelf CSP solver is used to automatically analyse the input data and provide the analysis results. CSP solvers search for a valid set of variable values that simultaneously satisfies all constraints. For example, $A+B>1$ is a CSP involving the integer variables A and B, both with a domain $\in [1..10]$. In this case, the solver would find ($A=2$, $B=2$) as a valid solution for the CSP.

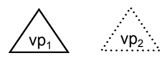
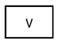
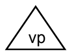
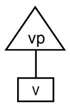
5.1 Mapping OVM+φ to CSP

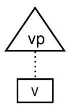
The mapping of an OVM+φ into CSP can differ depending on the concrete solver that is used later to solve the problem. In general, the mapping process goes through two main steps. First, the 3-tuple $\psi_{ovm} = (V_{ovm}, D_{ovm}, C_{ovm})$ is built, where the variable elements in the OVM become variables in V_{ovm} with their respective domains in D_{ovm}, and the variability and constraint dependencies in the OVM become constraints in the C_{ovm}. Second, the final mapping from an OVM+φ to a CSP is carried out by adding variables and constraints to the ψ_{ovm}, where the quality attributes become variables and the domain constraints become constraints, resulting in the 3-tuple $\psi_{ovm+\varphi} = (V_{ovm+\varphi}, D_{ovm+\varphi}, C_{ovm+\varphi})$. Next, we detail the complete mapping process.

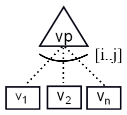
5.1.1 Building the ψ_{ovm}

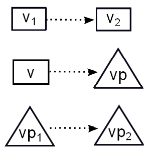
Concrete rules for mapping an OVM into a CSP are listed in Table 5. Also, the mapping of our RFW example of Figure 13 is presented. In this table we show the mapping of an OVM into a general CSP, which is independent of the solver to be used later to analyse the model. The mapping is very similar to the one used for feature models (Benavides et al, 2005), with the following differences:

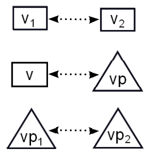
Table 5 Mapping OVM into Constraint Satisfaction Problem

		OVM	CSP mapping	Mapping example
Variation Point			$vp_1 \in [0,1] \land vp_2 \in [0,1]$	VP11:Sensor power ∈[0,1] ∧ VP10:Signs giving orders ∈ [0,1]
Variant			v ∈ [0,1]	V50:Low ∈ [0,1] ∧ V51:Medium ∈ [0,1] ∧ V52:High ∈ [0,1] ∧ V41:No overtaking ∈ [0,1] ∧ V44:End of Prohibitions ∈ [0,1]
Variation Point	MANDATORY		vp = 1	VP11:Sensor power = 1
Variability Dependency	MANDATORY		vp = v	

	OPTIONAL		if (vp = 0)
			v = 0
				if (VP10:Signs giving orders = 0)
				V41:No overtaking = 0
				
				if (VP10:Signs giving orders = 0)
				V44:End of Prohibitions = 0

	ALTERNATIVE		if (vp > 0)
			sum (v1, v2, ..., vn) in {i..j}
			else
			v1 = 0, v2 = 0, ..., vn = 0
				if (VP11:Sensor power > 0)
				sum (V50:Low ,V51:Medium, V52:High)∈ {1..1}
				else
				V50:Low = 0 ,V51:Medium = 0, V52:High = 0

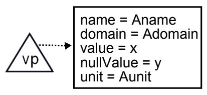
Constraint Dependency	REQUIRES		if (v1 > 0)
			v2 > 0
			if (v > 0)
			vp > 0
			if (vp1 > 0)
			vp2 > 0	

	EXCLUDES		if (v1 > 0)
			v2 = 0
			if (v > 0)
			vp = 0
			if (vp1 > 0)
			vp2 = 0	

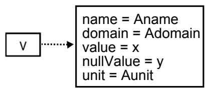
1. In the OVM model, there are two types of nodes, namely variation points and variants. These nodes differ from each other. Variation points are mandatory or optional. In feature models, all nodes in the diagram are features.
2. There is no constraint for a root node, since there is no root node in the OVM.
3. For each mandatory variation point, we add a constraint assigning value 1 to the correspondent variable.
4. Each alternative relationship is mapped to a constraint "$if(vp > 0)\ sum(v_1, v_2, \ldots, v_n)\ in\ \{m..m'\}\ else\ v_1 = 0 \land v_2 = 0 \land v_n = 0$", where vp is the variation point, $v_i \mid i \in [1 \ldots\ n]$ the

Table 6 Mapping OVM+φ into Constraint Satisfaction Problem

	OVM+φ		$\Psi_{ovm+\varphi}$	mapping example

Attributes related to variation point	 vp
	name = Aname
	domain = Adomain
	value = x
	nullValue = y
	unit = Aunit
		Domain	vp.Aname ∈ {Adomain}∪{y}	VP11:Sensor power.Range ∈ Integer ∪{-∞ }

		Constraint	If vp = 1 then
			vp.Aname = x
			else vp.Aname = y
				If VP11:Sensor power = 1 then
				VP11:Sensor power.Range = max(
				V50:Low.Range,
				V51:Medium.Range,
				V52:High.Range)
				else VP11:Sensor power.Range = - ∞

Attributes related to variants	 v
	name = Aname
	domain = Adomain
	value = x
	nullValue = y
	unit = Aunit
		Domain	v.Aname ∈ {Adomain}∪{y}	V50:Low.Range ∈ Integer ∪{-∞ } ∧
				V51:Medium.Range ∈ Integer ∪{-∞ } ∧
				V52:High.Range ∈ Integer ∪{-∞ }

		Constraint	If v = 1 then
			v.Aname = x
			else v.Aname = y
				(If V50:Low = 1 then
				V50:Low.Range = 70
				else V50:Low.Range = - ∞) ∧
				
				(If V51:Medium = 1 then
				V51:Medium.Range = 45
				else V51:Medium.Range = - ∞) ∧
				
				(If V52:High = 1 then
				V52:High.Range = 20
				else V52:High.Range = - ∞)

Global attribute	name = Aname
	domain = Adomain
	value = x
	nullValue = y
	unit = Aunit
			Aname ∈ {Adomain}	If VP11:Sensor power = 1 then
				VP11:Sensor power.Range = max(
				V50:Low.Range,
				V51:Medium.Range,
				V52:High.Range)
				else VP11:Sensor power.Range = - ∞

Domain constraint	domain constraint		constraint	(If V41:No over taking = 1 then VP11:Sensor power.Range >= 50) ∧ (If V44:End of prohibitions = 1 then VP11:Sensor power.Range >= 25)

set of optional variants in the relationship, and $[m \dots m'] \mid 0 \leq m \leq m' \leq n$ the cardinality. This mapping is similar to cardinality-based feature models.

5.1.2 Building the $\psi_{ovm+\varphi}$

After we have built the ψ_{ovm}, we add the variables corresponding to each attribute in the OVM+φ and the needed constraints, thus resulting in the $\psi_{ovm+\varphi}$. The general mapping rules and the mapping for the RFW example of Figure 13 are presented in Table 6. These mappings created following these steps:

1. Each attribute in the OVM+φ becomes a variable in $\psi_{ovm+\varphi}$. The domain of these variables are defined by the union of the domain interval specified to the corresponding attribute and its nullValue. Note that, when an attribute is global it is part of all products, therefore it does not need a neutral value.
2. The values of each attribute in the OVM+φ become constraints on a variable in $\psi_{ovm+\varphi}$.
3. The domain constraints in the OVM+φ become constraints in $\psi_{ovm+\varphi}$.

5.2 Defining operations for detecting anomalies as CSP primitives

The analysis method we propose is characterised by analysis operations that are applied to an OVM+φ. In this section, we define the three analysis operations we need to detect anomalies in the OVM+φ as CSP primitives.

Operation 1 (Void Model). Let $ovm+q$ be an OVM+φ specification, and $\psi_{ovm+\varphi}$ its equivalent CSP. Then, $ovm+q$ is void if there is no solution to $\psi_{ovm+\varphi}$.

$$void(\psi_{ovm+\varphi}) \Leftrightarrow |sol(\psi_{ovm+\varphi})| = 0$$

where $sol(\psi_{ovm+\varphi})$ is the set of solutions of $\psi_{ovm+\varphi}$.

Operation 2 (Dead Variable Element). Let $ovm+q$ be an OVM+φ specification and let $\psi_{ovm+\varphi}$ be its equivalent CSP of the form (V_{ovm+q}, D_{ovm+q}, C_{ovm+q}). The variable element ve $\in V_{ovm+q}$ *is dead if it does not belong to any solution of $\psi_{ovm+\varphi}$. It follows:*

$$isDead(\psi_{ovm+\varphi}, ve) \Leftrightarrow \forall s : sol(\psi_{ovm+\varphi}) \cdot ve \notin s$$

A variable element is false optional if it is modelled as optional, but it appears in all valid products. A variable element can be a variant or a variation point. A variation point vp is false optional if it is optional and is part of all solutions of $\psi_{ovm+\varphi}$. A variant v is false optional when it is part of all solutions of $\psi_{ovm+\varphi}$ and *i)* its parent relationship is optional or alternative, or *ii)* its parent is optional. Consider that parent(v) returns the parent of v, relationship() returns the type of relationship between a variant v and its parent, and vptype() returns whether the variation point is mandatory or optional. The false optional operation is defined as follows:

Operation 3 (False Optional (FO)). Let $ovm+q$ be an OVM+φ specification, $\psi_{ovm+\varphi}$ be its equivalent CSP of the form ($V_{ovm+\varphi}$, $D_{ovm+\varphi}$, $C_{ovm+\varphi}$), and $v, vp \in V_{ovm+\varphi}$. Then, vp is false optional if $isFalseOpt(\psi_{ovm+\varphi}, vp) = true$ and v is false optional if $isFalseOpt(\psi_{ovm+\varphi}, v) = true$. It follows:

$$isFalseOpt(\psi_{ovm+\varphi}, vp) \Leftrightarrow \forall s : sol(\psi_{ovm+\varphi}) \cdot vp \in s \wedge vptype(vp) = optional$$

$$isFalseOpt(\psi_{ovm+\varphi}, v) \Leftrightarrow \forall s : sol(\psi_{ovm+\varphi}) \cdot$$
$$(v \in s \wedge (relationship(parent(v)) = optional \vee alternative)) \vee (v \in s \wedge (vptype(parent(v)) = optional))$$

6 Analysing OVM+φ

Apart from analysis operations to detect anomalies in OVM+φ, we propose other analysis operations. Quality-aware analysis operations allow verifying whether a product or a set of products in a product line specification fulfil a given quality condition, at the same time that satisfy functional variability and domain constraints. This analysis aims at supporting the engineer decisions, but it can be performed by any stakeholder that wants to obtain information from the product line. Therefore, if an engineer of a product line has at his/her disposal an OVM+φ specification, he/she can make use of quality-aware analysis.

The analysis process introduced in Figure 15 is extended in order to consider quality conditions (see Figure 16). In this process, an operation has as input a target OVM+φ, may or may not have a partial configuration (a set of variants that must be in the products and

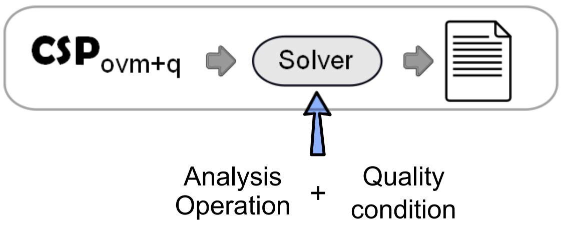


Fig. 16 Process for the automated analysis of OVM+φ with quality conditions

a set of variants that cannot be), and a quality condition. We define a quality condition as a statement of what is required as part of a product or a set of products regarding quality attributes. For example, the engineer of the RFW product line may want to analyse if it is possible to derive a particular product with development cost no higher than the assigned budget. The restrictions imposed by the engineer when expressed as quality conditions can be used to get the answer from the OVM+φ specification. For example, the quality condition defined by the engineer can be expressed as $TotalAccuracy < 10 \wedge TotalCost < 30$. As previously mentioned, our approach is automated by means of CSP. Therefore, our basic assumption is that quality conditions can be specified as a constraint on quality attributes, as shown in the following:

Definition 1 (Quality condition). Let $ovm+q$ be an OVM+φ specification, $\psi_{ovm+\varphi}$ be its equivalent CSP of the form (V_{ovm+q}, D_{ovm+q}, C_{ovm+q}). A quality condition q is a constraint on one or more attributes $\in V_{ovm+q}$.

6.1 Satisfiability

As previously mentioned, the engineer of the RFW product line may want to verify if it is possible to configure a product that satisfies a given quality condition. We propose an analysis operation, namely satisfiability, to verify whether a product or a set of products in OVM+φ fulfil a given quality condition, at the same time that satisfy functional variability and domain constraints. A product line satisfies a quality condition if there is at least one product that satisfies all constraints defined in both, OVM+φ and quality condition. Let us, for example, consider the quality condition previously defined $TotalAccuracy < 10 \wedge TotalCost < 30$. Then, the product line satisfies this quality condition if there is at least one solution to $\psi_{ovm+\varphi} \wedge TotalAccuracy < 10 \wedge TotalCost < 30$, as follows:

$$Satisfies(\psi_{ovm+\varphi}, TotalAccuracy < 10 \wedge TotalCost < 30) \Leftrightarrow$$
$$|sol(\psi_{ovm+\varphi} \wedge TotalAccuracy < 10 \wedge TotalCost < 30)| > 0$$

Additionally, taking into account the needs of the stakeholders, the engineers can verify whether some partial configuration (i.e., a set of variants) satisfies the variability expressed by the OVM+φ and a quality condition, as well. Therefore, if necessary, the engineer can try to achieve the required configuration by relaxing or removing relationships in the variability model.

A partial configuration is of the form $\{Se, Re\}$, where Se is the set of variants to be selected, and Re is the set of variants to be removed from the configuration, such as $\forall\ v_i \in Se \rightarrow v_i = 1$ and $\forall\ v_i \in Re \rightarrow v_i = 0$. For example, the RFW engineer wants to verify if it is possible to derive a product by combining the most powerful sensor, the GPS positioning system, and the medium-class car. Thus, this partial configuration is expressed by the set $\{\{$`V53:GPS`, `V52:High`, `V1:Medium-class car`$\}\{\}\}$. Therefore, the OVM+φ satisfies the engineers quality condition and simultaneously offer the desired partial configuration if and only if there is at least one solution to $\psi_{ovm+\varphi} \wedge TotalAccuracy < 10 \wedge TotalCost < 30 \wedge$ $\{$`V53:GPS`, `V52:High`, `V1:Medium-class car`$\}\{\}$, as shown in the following:

$$\begin{gathered} Satisfies(\psi_{ovm+\varphi}, TotalAccuracy < 10 \wedge TotalCost < 30, \\ \{\{V53:GPS, V52:High, V1:Medium-classcar\}\{\}\}) \Leftrightarrow \\ |sol(\psi_{ovm+\varphi} \wedge TotalAccuracy < 10 \wedge TotalCost < 30) \wedge \\ (\{\{V53:GPS, V52:High, V1:Medium-classcar\}\{\}\})| > 0 \end{gathered}$$

6.2 Optimal product

We refer to optimal product as the product that satisfies OVM+φ, and also minimises or maximises a given objective function. When relating quality information to OVM we are able to ask for an optimal product, since the objective function takes into account values of attributes. The product line engineer may want to verify, for example, which product of the set of products consumes less memory, or even to find the product with the lowest development cost. Therefore, finding the optimal solution, as opposed to any possible solution, would be helpful for making quality-aware decisions. Hence, to find the optimal solution, we can associate an objective function with the CSP. Then, the solver has to find solutions that maximise or minimise the specified objective function that satisfies all the constraints. A possible objective function would be $O = TotalCost$, such that the sought solution is rendered by minimising O, as follows:

$$Cheapest product(s) = min(CSP_{ovm+q}, TotalCost)$$

6.2.1 Optimal product with quality condition

The engineer of a product line may want to verify which is the optimal product that satisfies some quality condition and a desired partial configuration. For example, which is the cheapest product that offers $TotalAccuracy < 10 \wedge TotalCost < 30$ and has the variants `V53:GPS`, `V52:High`, and `V1:Medium-class car`? To find this optimal solution, we first filter the model by adding a quality condition (Φ) and a partial configuration (PC), which were introduced in Section 6.1. Afterwards, we define the objective function. In this case, the objective function is the global attribute $TotalCost$, which has been defined as follows (see Table 4 for a complete definition):

$$\sum_{j=1}^{k} vp_j.Cost$$

Finally, we define that the optimal products (P_{opt}) minimise the $TotalCost$, as shown in the following:

$$\begin{aligned} \Phi &= TotalAccuracy < 10 \wedge TotalCost < 30 \\ PC &= \{\{V53 : GPS, V52 : High, V1 : Medium - classcar\}\{\}\} \\ filter &= CSP_{ovm+q} \wedge \Phi \wedge PC \\ O &= TotalCost \\ P_{opt} &= min(filter, O) \end{aligned}$$

6.2.2 Most representative product

The optimization operation can be used to find the most representative product(s) of a product line, which could be used to support evaluation strategies. Assessing all possible products of the product line is impracticable due to the usually very large number of products in a product line. Therefore, strategies to decide which products should be checked are needed.

There may be different ways of implementing the *Most Representative Products (MRP)* operation. In this article, we consider the MRP those that have variants involved in a larger number of products, however there are other possibilities (e.g., the most expensive ones or those that have the largest number of variants). Therefore, the MRP operation is defined as the product(s) of the software product line that maximise(s) the commonality degree. The commonality degree of a product is determined by the sum of the commonality of its variants and variation points. This commonality represents the percentage of products where the variable element appears, e.g., if there are 10 possible products and a variant *v* appears in 5 products, the commonality of *v* is 50%. After the most representative product has been found, any evaluation technique employed in single-systems can be applied to evaluate its quality. This operation is defined as follows:

Operation 4 (Most Representative Products (MRP)). Let $ovm+q$ be an OVM+φ specification and $v \in V_{ovm+\varphi}$. Commonality Degree of v is the percentage of products (solutions) in which v is included. O is the summation of commonality degree of variants, and MRP is the product(s) that maximise(s) O.

$$CommonalityDegree(v) = \frac{|sol(\psi_{ovm+q} \wedge v)|}{|sol(\psi_{ovm+q})|}$$

$$O = \sum_{i=1}^{k} CommonalityDegree(v_i), where\ k = |V_{ovm+q}|$$

$$MRP(\psi_{ovm+\varphi}) = max(\psi_{ovm+\varphi}, O)$$

Note that, the MRP operation is defined based on an optimisation function represented by *O*. This function can be defined according to the user needs and is is quality-aware. For example, *O* could be defined as the *sum* of the attributes `Cost` and therefore MRP would return the most expensive product(s).

7 Implementing the approach

7.1 FaMa-OVM

We have developed FaMa-OVM, which is a prototype tool to implement our approach. FaMa-OVM receives as input an OVM+φ specification and provides support to quality-aware analysis process. It allows detecting anomalies in an OVM+φ specification, as well

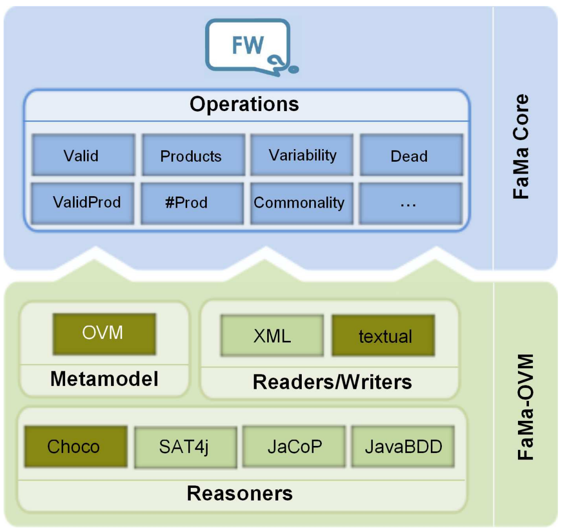


Fig. 17 FaMa-OVM, an extension of FaMa-FW

as verifying quality conditions, optimal products, and the most representative product(s). The tool was implemented based on FaMa-Framework (FaMa-FW) (Trinidad et al, 2008b), which is an open source Java framework designed to facilitate the development of analysis tools for diverse variability modelling languages. FaMa-FW provides a number of extension points to plug in new components, such as metamodels, readers/writers and reasoners. Figure 17 shows an overview of FaMa-OVM components. The dark components are the extensions of the original framework. In the following, we report on those components we have implemented:

The *OVM+φ metamodel* implements the description of the different variable elements, and the rules that constraint the combination of these elements. Furthermore, it describes the attributes and their relationship with variable elements, as well as the constraints on attributes.

The *OVM+φ reader* implements a reader to an OVM+φ textual format, which we have defined for representing an OVM+φ specification. Figure 18 shows a single textual format for the examples in Figures 12 and 13. This textual format consists of four main parts, namely: *Relationships*, *Attributes*, *Global Attributes* and *Constraints*. *Relationships* specifies the variability dependencies between variation points and variants. *Attributes* specifies basic and derived attributes. *Global Attributes* specifies global attributes. *Constraints* specifies excludes and requires relationships, and quality conditions.

Attributes are defined as follows:

$< name > : < domain >, < value >, < nullValue >;$

The terms are separated by commas, and lines are finished with semicolon. When the value of an attribute is a function, a semicolon after $< value >$ is used.

The *OVM+Q reasoner* implements the solver by using Choco (Laburthe et al, accessed November 2010). A CSP solver was used because it offers the possibility to work with numerical values, such as integer, which allows dealing with attributes, enabling it to maximise or minimise values.

```

%Relationships
 VP12PositioningSystem : [1,2]{V53GPS V54Galileo};
 VP13Antenna : [1,1]{V55Small V56Medium V57Big};
 VP11SensorPower: [1,1]{V50Low V51Medium V52High};
 [VP10SignsGivingOrders] : [V41NoOvertaking] [V44EndOfProhibitions];

%Attributes
 V53GPS.Accuracy: Integer[1 to 10], 8, INF;
 V54Galileo.Accuracy: Integer[1 to 10], 4, INF;
 VP12PositioningSystem.Accuracy: Integer[1 to 10], min(V53GPS.
     Accuracy,V54Galileo.Accuracy);,INF;

 V55Small.AccuracyFactor: Integer [1 to 4], 4, MINF;
 V56Medium.AccuracyFactor: Integer [1 to 4], 3, MINF;
 V57Big.AccuracyFactor: Integer [1 to 4], 1, MINF;
 VP13Antenna.AccuracyFactor: Integer [1 to 4], max(V55Small.
     AccuracyFactor, V56Medium.AccuracyFactor,V57Big.AccuracyFactor)
     ;,MINF;

 V50Low.Range: Integer [20 to 30], 20, MINF;
 V51Medium.Range: Integer [45 to 60], 45, MINF;
 V52High.Range: Integer [70 to 100], 70, MINF;
 VP11SensorPower.Range: Integer [1 to 100], max(V50Low.Range,
     V51Medium.Range,V52High.Range);,MINF;

%GlobalAttributes
 TotalAccuracy: Integer [1 to 40], VP12PositioningSystem.Accuracy *
     VP13Antenna.AccuracyFactor;, 0;

%Constraints
 V53GPS EXCLUDES V55Small;
 V41NoOvertaking IMPLIES VP11SensorPower.Range >= 50;
 V44EndOfProhibitions IMPLIES VP11SensorPower.Range >= 25;
```

Fig. 18 FaMa-OVM textual format

As we have used Choco solver, all variables in the CSP must belong to a finite domain, which implies that attributes must have a finite domain. Due to this limitation, functions can only involve integer values. Consequently, we were unable to use real numbers as we intended. Subsequently, real numbers were mapped to integers. The values of the attributes `V55Small.AccuracyFactor`, `V56Medium.AccuracyFactor`, `V57Big.AccuracyFactor` were mapped from [1.5,1,0.25] to [4,3,1], respectively.

7.2 Analysis results

In this section, we present the analysis results we have obtained with FaMa-OVM. Our tool provides support to quality-aware analysis of OVM+φ. It is worth mentioning that we have no intention of providing an industrial tool support for such quality-aware analysis, but offer a proof of concepts of our approach.

The analysis of variability models with attributes is a complex problem. When specifying quality attributes we have defined a certain domain for them. The domain sets the limits of the attribute values. We address the analysis problem as a CSP, then the higher

		RFW model†±	Excerpt of RFW ‡ ∓
Detect anomalies	**Void**	False	False
	Dead	V38Bend V39Traffic queues V45Yield V50Low V51Medium V55Small V56Medium	None
	False Optional	VP7Other signs VP8Prohibition signs VP9Warning signs VP10Signs given orders V41No overtaking V57Big	None
Satisfiability	**Satisfies(QC)**	False	True
	Satisfies(QC+PC)	False	True

† $QC = TotalAccuracy < 10 \wedge TotalCosts < 30$
‡ $QC = TotalAccuracy < 10$
± $PC = \{\{V53 : GPS, V52 : High, V1 : Medium - ClassCar\}, \{\}\}$
∓ $PC = \{\{V53 : GPS, V52 : High, \{\}\}$

Table 7 Results for some of the analysis operations

the range of the domain is, the more complex the problem becomes. In our implementation, when mapping the RFW example to a CSP, we have replaced the domain range of attributes as much as possible in order to reduce the problem complexity. For example, in the case of `V11.Latency`, we have replaced the range *Integer[200..800]* by *[400]*, and in the case of `VP4Behavior at warning signs.Latency`, we have replaced *Integer[200..800]* by $[350, 400, 500]$. An effort was made to preserve consistency amongst the assigned values.

We have employed FaMa-OVM to execute all the operations defined in this article. We have analysed two models: (i) the *RFW model*, which represents the RFW product line with all attributes and domain constraints that were defined through this article, and (ii) the *Excerpt of RFW* model, which is depicted in Figure 8. The textual OVM for the RFW model can be found in the complementary material provided at the end of this article.

We were able to find solutions for five operations when applied to both models, as can be seen in Table 7. We have verified that none of the models is void, but other anomalies where detected. The excerpt model does not have any anomaly, however the RFW model has seven dead elements and six false optional. These anomalies were caused by the wrong use of constraints by the product line engineer. We were able to determine that the constraints that involves $TotalAccuracy <= 10$ are causing some of the dead elements as well as the false optional.

Furthermore, we were able to verify that the RFW product line does not satisfy the quality condition $TotalAcurancy < 10 \wedge TotalCost < 30$. When we have relaxed this quality condition, by changing values to $TotalAccuracy < 30 \wedge TotalCost <= 50$, we have found that there are products which satisfy the relaxed quality condition. As can be seen in Table 7, we have also analysed the excerpt model and seen that it satisfies another quality condition, namely $TotalAcurancy < 10$. The number of products found in the excerpt model when no quality conditions were defined is 70, but when analysing it using the quality condition $TotalAcurancy < 10$, this number was reduced to 30.

In addition, we have analysed whether the RFW product line satisfies the quality condition $TotalAcurancy < 10 \wedge TotalCost < 30$ associated with the partial configuration $\{\{V53GPS, V52High, V1Medium-ClassCar\},\{\}\}$. As shown in Table 7, we have found that there are no products which have `V53GPS`, `V52High`, and `V1Medium-ClassCar`, and satisfy such quality condition. In the case of the excerpt model, we have verified that there are products which have `V53GPS` and `V52High`, and satisfy the quality condition $TotalAcurancy <$ 10. When we have associated the partial configuration with the quality condition and verified satisfiability, the number of products found in the excerpt model was reduced to 10.

There are operations that need to compute all possible solutions beforehand to be able to find a solution, such as the optimal product and the MRP. Therefore the problem to be solved is more complex. We have observed that for the RFW model these two operations have taken much more time. The Choco solver was not able to find the most representative and the optimal products in a reasonable time; however, it found a solution to these two operations when analysing the excerpt model. In this model we were able to find the most accurate products. For this purpose we have defined that the optimal products minimise the *TotalAccuracy* global attribute. Thus, we have found that 20 products are able to provide the minimum accuracy, which is 4. In the following, we present the 20 most accurate products.

```
Optimal product 1 = {VP12PositioningSystem,V54Galileo,VP13Antenna,V57Big,
                     VP11SensorPower,V52High}

Optimal product 2 = {VP12PositioningSystem,V54Galileo,VP13Antenna,V57Big,
                     VP11SensorPower,V52High,VP10SignsGivingOrders}

Optimal product 3 = {VP12PositioningSystem,V54Galileo,VP13Antenna,V57Big,
                     VP11SensorPower,V52High,VP10SignsGivingOrders,
                     V44EndOfProhibitions}

Optimal product 4 = {VP12PositioningSystem,V54Galileo,VP13Antenna,V57Big,
                     VP11SensorPower,V52High,VP10SignsGivingOrders,
                     V41NoOvertaking}

Optimal product 5 = {VP12PositioningSystem,V54Galileo,VP13Antenna,V57Big,
                     VP11SensorPower,V52High,VP10SignsGivingOrders,
                     V41NoOvertaking,V44EndOfProhibitions}

Optimal product 6 = {VP12PositioningSystem,V54Galileo,VP13Antenna,V57Big,
                     VP11SensorPower,V51Medium}

Optimal product 7 = {VP12PositioningSystem,V54Galileo,VP13Antenna,V57Big,
                     VP11SensorPower,V51Medium,VP10SignsGivingOrders}

Optimal product 8 = {VP12PositioningSystem,V54Galileo,VP13Antenna,V57Big,
                     VP11SensorPower,V51Medium,VP10SignsGivingOrders,
                     V44EndOfProhibitions}

Optimal product 9 = {VP12PositioningSystem,V54Galileo,VP13Antenna,V57Big,
                     VP11SensorPower,V50Low}

Optimal product 10 = {VP12PositioningSystem,V54Galileo,VP13Antenna,V57Big,
                      VP11SensorPower,V50Low,VP10SignsGivingOrders}

Optimal product 11 = {VP12PositioningSystem,V53GPS,V54Galileo,VP13Antenna,
                      V57Big,VP11SensorPower,V52High}

Optimal product 12 = {VP12PositioningSystem,V53GPS,V54Galileo,VP13Antenna,
                      V57Big,VP11SensorPower,V52High,VP10SignsGivingOrders}
```

```
Optimal product 13 = {VP12PositioningSystem,V53GPS,V54Galileo,VP13Antenna,
                      V57Big,VP11SensorPower,V52High,VP10SignsGivingOrders,
                      V44EndOfProhibitions}

Optimal product 14 = {VP12PositioningSystem,V53GPS,V54Galileo,VP13Antenna,
                      V57Big,VP11SensorPower,V52High,VP10SignsGivingOrders,
                      V41NoOvertaking}

Optimal product 15 = {VP12PositioningSystem,V53GPS,V54Galileo,VP13Antenna,
                      V57Big,VP11SensorPower,V52High,VP10SignsGivingOrders,
                      V41NoOvertaking,V44EndOfProhibitions}

Optimal product 16 = {VP12PositioningSystem,V53GPS,V54Galileo,VP13Antenna,
                      V57Big,VP11SensorPower,V51Medium}

Optimal product 17 = {VP12PositioningSystem,V53GPS,V54Galileo,VP13Antenna,
                      V57Big,VP11SensorPower,V51Medium,VP10SignsGivingOrders}

Optimal product 18 = {VP12PositioningSystem,V53GPS,V54Galileo,VP13Antenna,
                      V57Big,VP11SensorPower,V51Medium,VP10SignsGivingOrders,
                      V44EndOfProhibitions}

Optimal product 19 = {VP12PositioningSystem,V53GPS,V54Galileo,VP13Antenna,
                      V57Big,VP11SensorPower,V50Low}

Optimal product 20 = {VP12PositioningSystem,V53GPS,V54Galileo,VP13Antenna,
                      V57Big,VP11SensorPower,V50Low,VP10SignsGivingOrders}
```

Furthermore, we have analysed the excerpt model by executing the optimal operation associated with the quality condition $TotalAcurancy < 10$ and the partial configuration $\{\{V53GPS, V52High\}, \{\}\}$, and still using $TotalAccuracy$ as the objective function. In this case, five products were found as the most accurate products, namely:

```
Optimal product 1 = {VP12PositioningSystem,V53GPS,V54Galileo,VP13Antenna,V57Big,
                     VP11SensorPower,V52High}

Optimal product 2 = {VP12PositioningSystem,V53GPS,V54Galileo,VP13Antenna,V57Big,
                     VP11SensorPower,V52High,VP10SignsGivingOrders}

Optimal product 3 = {VP12PositioningSystem,V53GPS,V54Galileo,VP13Antenna,V57Big,
                     VP11SensorPower, V52High,VP10SignsGivingOrders,
                     V44EndOfProhibitions}

Optimal product 4 = {VP12PositioningSystem,V53GPS,V54Galileo,VP13Antenna,V57Big,
                     VP11SensorPower,V52High,VP10SignsGivingOrders,V41NoOvertaking}

Optimal product 5 = {VP12PositioningSystem,V53GPS,V54Galileo,VP13Antenna,V57Big,
                     VP11SensorPower,V52High,VP10SignsGivingOrders,V41NoOvertaking,
                     V44EndOfProhibitions}
```

In addition, we present the two most representative products we have obtained with our tool.

```
Most representative product 1 = {VP12PositioningSystem,V53GPS,V54Galileo,
                                 VP13Antenna,V56Medium,VP11SensorPower,
                                 V52High,VP10SignsGivingOrders,
                                 V41NoOvertaking,V44EndOfProhibitions}
```

```
Most representative product 2 = {VP12PositioningSystem,V53GPS,V54Galileo,
                                  VP13Antenna,V57Big,VP11SensorPower,
                                  V52High,VP10SignsGivingOrders,
                                  V41NoOvertaking,V44EndOfProhibitions}
```

We ran the experiments on a computer that is equipped with a Dual core AMD Opteron 1218 processor running at 2.6GHz, 2GB of RAM, Ubuntu 10.10 with Kernel 2.6.35.28, and the 1.6.0 version of the Java Runtime Environment.

8 Related work

A number of research efforts have been made to capture quality attributes and their relationship with functional features (Montagud and Abrahão, 2009). The addition of quality information to variability models has mostly been proposed for feature model approaches, however, in the context of OVM, it has not been explored in the literature before. The relationship between feature models and some additional information was already suggested by (Kang et al, 1990) in the seminal feature model proposal called FODA. In this work, the authors contemplate the addition of feature attributes with quantified values. They also introduce the need to define relationships between features and attributes. Later, (Kang et al, 1998) make an explicit reference to non-functional features, which they define as a kind of feature that characterises functional features. Other authors have also proposed the extension of feature models with so-called feature attributes (Batory, 2005; Batory et al, 2006; Benavides et al, 2005; Czarnecki et al, 2005). Feature models and OVMs differ in the way they relate to quality information. Feature models can be directly annotated with features, whereas in OVM, the quality information must be in a separate model. As OVM documents variability realised in the base models and this variability is orthogonal to all base models, it cannot be annotated with attributes.

There are several proposals providing automated analysis of basic or cardinality-based feature models (Benavides et al, 2010). Several analysis operations on feature models and also automated support for them were proposed. These approaches can be classified in four different groups, according to the logic paradigm or method used to provide automated support: propositional logic, constraint programming, description logic, and ad-hoc algorithms. Most of the approaches that deal with quality information propose the use of constraint programming for automating, since it allows dealing with integer variables. However, feature models extended with attributes, where numerical values are included, have not received much attention (Benavides et al, 2010).

Benavides et al (2005) propose *extended feature models* and the automated analysis of such models by using CSP. Although in this approach the authors provide a way to add attributes to features, they do not provide much detail about the values of the attributes, as acknowledged in Benavides et al (2010). In this work they only present a simple and high-level example were they illustrate a possible mapping from feature attributes to CSP. The values of the attributes are ranges and the only function used to calculate derived attributes is the *addition*. There is no global attribute and relationships between attributes are hierarchically organised in the feature tree, in other words, the relationship between attributes exists only between a parent and a child, and they do not provide support to constraint on attributes. In addition, the authors propose a number of analysis operations on the extended feature model, however, they do not provide quality-aware analysis.

Karataş et al (2010) propose a quality language to express extended feature model. They address feature-attribute and attribute-attribute constraints. In addition, they provide a mapping from an extended feature model to a CSP. We were inspired by their ideas to define a quality information language. It is to note that is would be easy to introduce their results in our proposal. Although they offer a detailed quality language, they neither cover derived and nor global attributes. In addition, functions as values for the attributes are not allowed. Furthermore, the proposed mapping is different from ours. In their proposal, they also do not have null values for the attributes, so, to map each constraint into a CSP, they need to add a constraint indicating that the features involved in the constraint must be selected. In other words, in their approach, the value of an attribute is relevant to a constraint only if the feature it belongs to is included in the product.

White et al (2009) present a method called Filtered Cartesian Flattening to solve the problem of optimally selecting a set of features that simultaneously satisfy a number of resource constraints. They apply several existing algorithms to this problem, which perform much faster and offer an approximate solution. We consider this work complementary to ours, and their research results could be applied to our problem of finding an optimal product.

Other authors Tun et al (2009); Bagheri et al (2010) have looked at more qualitative means of evaluating quality constraints based on goal-oriented analysis. In (Tun et al, 2009), the authors separate feature descriptions into feature models relating to the requirements, the problem world context, and the specifications. Once requirements are selected for a desired product, one or more products that satisfy the requirements and the quantitative quality constraints are generated. This article resembles our research regarding the analysis of quality information in variability models. They use quality attributes and quality constraints to express technically known properties of the system. However, both approaches differ with regard to the way, quality attributes are defined. In contrast to our approach, they define the value of derived attributes as a hierarchical function, which means that the value of the attribute of a parent feature is calculated as a function of the child feature's values. This hierarchy limits the relationship between attributes. In addition, they provide a way to configure, from a feature model, solutions that satisfy quality requirements as well as quality constraints. In our work, on the other hand, we propose a number of quality-aware analysis operations to analyse OVM. Bagheri et al (2010) take into consideration the stakeholders' desired quality attributes (called *soft constraints*) during a feature model configuration, and use a fuzzy propositional language for the analysis. They provide an interactive feature model configuration process, where they annotate features with high-level abstract objectives. By using fuzzy form, they express how features contribute to satisfy these objectives. This work focuses on related feature models with strategic objective of the stakeholders to find the solution that best fits stakeholders' desire.

There is also research that is based on other variability model techniques. Sinnema et al (2004) propose COVAMOF, which is a variability modelling framework for modelling variabilities and constraints on quality properties. In COVAMOF quality attributes are expressed as dependencies related to variation points. These dependencies have, among other things, a function that determines their values, depending on the selected variants, and a constraint on these values. The value of the dependencies can represent formal or informal knowledge. The former is represented using algebraic expressions, and the latter can contain or refer to documented knowledge (e.g., HTML documents). Dhungana et al (2010) propose the decision-oriented variability modelling language DOPLERVML as part of the DOPLER tool suite (Dhungana et al, 2007). In this language, the decisions represent the variation points in a variability model and the assets describe the reusable artefacts and their dependencies. In contrast with our proposal, in COVAMOF attributes are only related to variation points, but not with variants, thus direct impact from one variant to another cannot be specified.

In DOPLERVML attributes are related to assets and not to the variability model, besides constraints on attributes are not considered by the authors. Furthermore, both proposals do not provide automated analysis.

To the best of our knowledge, only Metzger et al (2007) have partially explored the automated analysis of OVM. As part of their work, they propose an indirect way to automatically analyse OVMs. First, they transform an OVM into a Varied Feature Diagram (VFD$^+$), which is a formal "back-end" language used to define semantics and automating analysis, and in doing so, they reuse the semantics of analysis operations on VFD$^+$. To carry out this transformation, they provide an ad–hoc algorithm. Second, they map the VFD$^+$ to a propositional formula and then automatically analyse the OVM by means of the solver SAT4j (Berre and Parrain, accessed November 2010). In contrast to our proposal, they do not address quality information.

9 Discussions and conclusions

In this article, we have presented an approach for quality-aware analysis in software product lines using OVM to represent variability. To provide a method for quality-aware analysis, we follow three steps. First, we have presented a way to associate quality information with OVM (referred to as OVM+φ). Second, we have proposed a number of analysis operations to verify OVM+φ. Third, we have presented a mapping from OVM+φ to a constraint satisfaction problem and use Choco, an off-the-shelf constraint programming solver, to automatically perform the analysis tasks. To illustrate the feasibility of our approach, we have used a product line example from the automotive domain, which was created in a national project by a leading car company. Besides, we have introduced FaMa-OVM, which is a prototypical tool developed as a proof of concepts of our approach. We were able to identify void models, dead and false optional elements, and check whether the product line example satisfies quality conditions. With FaMa-OVM results, we have determined that the addition of quality information to the analysis process highly increases the complexity of the problem when trying to find an optimal solution or the most representative product, as we have discussed in Section 7.2. We believe that it is important to work in collaboration with other research areas (e.g., Constraint Programming, Artificial Intelligence), in order to find other techniques that could better solve this problem.

In our approach we consider $1:1$ relationships between OVM and configuration model elements. However, these relationships could be extended to be of $1:N$ with only minor changes. These changes concern the mapping from OVM+φ to a CSP. In $1:1$ relationships, we can omit the configuration model in the mapping process and then each relationship between an OVM element and a quality attribute becomes a variant in the CSP. If $1:N$ is allowed, the configuration model cannot be omitted. Consequently, the mapping is changed and each relationship involving an OVM element, a configuration model element, and a quality attribute become a variable in the CSP. For the sake of simplicity we have not addressed $1:N$ relationships, although our approach is theoretically applicable.

In our approach, a quality attribute can be composed of values of other attributes, such as *TotalCost* and *TotalMemory*. However, the compositionality of certain attributes, such as security, usability, and performance is not obvious and often hard to define. In our approach, we deal with attributes that are technically known and that can be composed of means of functions on individual values. In addition, in our proposal, an attribute can be involved in many functions, yet these functions must allow a common neutral value. For example, if the neutral value of *GPS.Cost* is zero, then it cannot be involved simultaneously in an addition and a multiplication operation. Therefore, a limitation of our approach is that the same attribute can only be involved in multiple functions when it is possible to find a common neutral value for these functions.

There may be some cases in which the engineer wants to postpone the decision about the value to be assigned to an attribute, for example, the attribute *Cost* of a given functionality. In these cases, it would be helpful if a range and not a specific value could be specified for the value of the attributes related to variants. The use of ranges would slightly vary our approach, since we limit attribute values to concrete values or functions, however it would be possible to extend our proposal to support it. In fact, in our tool, ranges can be specified. One of the consequences of adding ranges as values for attributes is that the number of products could be increased and not reduced, as we have mentioned in Section 1. For example, it would be possible to have different products with the same functionalities but different levels for attributes. In our approach, the products differ only by functionalities.

Regarding our tool, we emphasise that it is based on a framework for the analysis of feature models, which is a research area that has been explored. We noted that this framework simplified the development, since we did not have to start from scratch. We have used a CSP solver to implement the analysis, Choco, to be precise. In a CSP, all variables belong to a finite domain, which implies that attributes must have a domain. Due to limitations in Choco, functions can only involve integer values. Consequently, we could not use real numbers as we intended, and real numbers were mapped to integers.

In this article, we have proposed a technique to support quality-aware analysis in SPLE with OVM. In this analysis, there can be different users of the different operations with different quality conditions. Therefore, the association between quality conditions and stakeholders would be determined by the user of the operation. For instance, minimising the development cost is important to the product provider, whereas accuracy of positioning system is far more interesting to the product customer.

In our experience, we have observed that although feature models and OVM are similar, both the specification of quality information in OVM and the implementation of our tool were not straightforward. This is due to their differences in structure and the way they relate to quality information. Furthermore, the addition of quality information to the analysis process highly increases the complexity of the problem.

The CSP solver we have used is not well prepared to solve such a complex problem as the quality-aware analysis. Our proposal provides a way to theoretically solve this problem, however when implementing FaMa-OVM to provide automated support for our approach, it was demonstrated that this problem cannot be solved within a reasonable time. Thus, we consider that it is important to investigate the nature of problems of the automated analysis of variability models associated with attributes.

Currently the FaMa-OVM textual format is created manually. To facilitate this task, we are implementing a textual editor using the Xtext framework (Foundation, accessed April 2011). In addition, we are working on the integration of FaMa-OVM with an OVM editor in order to offer a visual editor from where it is possible to execute quality-aware analysis. The visual OVM editor is included in the REMiDEMM (Requirements Engineering and Management in Domain Engineering with Multi-Model Interaction) case tool, which was presented in (Heuer et al, 2010).

Complementary material

The FaMa-OVM tool and the RFW textual format used in our evaluation are available at `http://www.lsi.us.es/~dbc/material/SofQualJ11`.

Acknowledgements We would like to thank Silvia Abrahão and Isidro Ramos for their helpful comments in earlier versions of this article. We also would like to thank José Galindo for his work on implementing FaMa-OVM tool. This work has been partially supported by the European Commission (FEDER) and Spanish Government under CICYT project SETI (TIN2009-07366), by the Andalusian Government under ISABEL (TIC-2533) and THEOS (TIC-5906) projects, by Evangelischer Entwicklungsdienst e.V. (EED) and by the DFG under grant PO 607/2-1 IST-SPL.

References

Bagheri E, Di Noia T, Ragone A, Gasevic D (2010) Configuring software product line feature models based on stakeholders' soft and hard requirements. In: Proceedings of the 14th international conference on Software product lines, Springer-Verlag, Berlin, Heidelberg, SPLC'10, pp 16–31

Batory D (2005) Feature models, grammars, and propositional formulas. In: 9th International Software Product Line Conference, Springer–Verlag, LNCS, vol 3714, pp 7–20

Batory D, Benavides D, Ruiz-Cortés A (2006) Automated analysis of feature models: Challenges ahead. Communications of the ACM 49(12):45–47

Benavides D, Trinidad P, Ruiz-Cortés A (2005) Automated reasoning on feature models. In: 17th Int. Conf. Advanced Information Systems Engineering, Springer–Verlag, LNCS, vol 3520, pp 491–503

Benavides D, Segura S, Ruiz-Cortés A (2010) Automated analysis of feature models 20 years later: A literature review. Information Systems 35(6):615 – 636

Berre DL, Parrain A (accessed November 2010) Sat4j solver. http://sat4j.org/

Chen L, Babar MA, Ali N (2009) Variability management in software product lines: A systematic review. In: 13th Intl. Software Product Line Conference, Carnegie Mellon University - Pittsburgh, PA, USA, pp 81–90

Czarnecki K, Helsen S, Eisenecker U (2005) Formalizing cardinality-based feature models and their specialization. Software Process: Improvement and Practice 10(1):7–29

Dhungana D, Rabiser R, Grünbacher P, Neumayer T (2007) Integrated tool support for software product line engineering. In: 22nd IEEE/ACM International Conference on Automated Software Engineering, ACM, New York, NY, USA, pp 533–534

Dhungana D, Heymans P, Rabiser R (2010) A formal semantics for decision-oriented variability modeling with dopler. In: Fourth International Workshop on Variability Modelling of Software–Intensive Systems, pp 29–35

Felfernig A, Friedrich GE, Jannach D (2000) UML as domain specific language for the construction of Knowledge-Based configuration systems. International Journal of Software Engineering and Knowledge Engineering (IJSEKE) 10(4):449–469

Finkel R, O'Sullivan B (2011) Reasoning about conditional constraint specification problems and feature models. Artificial Intelligence for Engineering Design, Analysis and Manufacturing 25(Special Issue 02):163–174

Foundation E (accessed April 2011) Xtext - language development framework. http://www.eclipse.org/Xtext/

Garcia F, Bertoa M, Calero C, Vallecillo A, Ruiz F, Piattini M, Genero M (2006) Towards a consistent terminology for software measurement. Information and Software Technology 48(8):631–644

Heuer A, Lauenroth K, Müller M, Scheele JN (2010) Towards effective visual modeling of complex software product lines. In: Proceedings of the 3rd International Workshop on Visualisation in Software Product Line Engineering (VISPLE) in Proceedings of the 14th International Software Product Line Conference Volume 2, pp 229–237

Kang K, Cohen S, Hess J, Novak W, Peterson S (1990) Feature–Oriented Domain Analysis (FODA) Feasibility Study. Tech. Rep. CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University

Kang KC, Kim S, Lee J, Kim K, Shin E, Huh M (1998) FORM: A feature–oriented reuse method with domain–specific reference architectures. Annals of Software Engineering 5(1):143–168

Karataş A, Oğuztüzün H, Doğru A (2010) Mapping extended feature models to constraint logic programming over finite domains. In: Bosch J, Lee J (eds) Software Product Lines: Going Beyond, LNCS, vol 6287, Springer Berlin / Heidelberg, pp 286–299

Laburthe F, Jussien N, Rochart G, Cambazard H, Prud'homme C, Malapert A, Menana J (accessed November 2010) Choco solver. http://choco.emn.fr/

Metzger A, Pohl K (2007) Variability management in software product line engineering. In: 29th International Conference on Software Engineering (ICSE Companion), IEEE Computer Society, pp 186–187

Metzger A, Pohl K, Heymans P, Schobbens P, Saval G (2007) Disambiguating the documentation of variability in software product lines: A separation of concerns, formalization and automated analysis. In: 15th Intl. Requirements Engineering Conference, pp 243–253

Montagud S, Abrahão S (2009) Gathering current knowledge about quality evaluation in software product lines. In: SPLC '09: Proceedings of the 13th International Software Product Line Conference, Carnegie Mellon University, Pittsburgh, PA, USA, pp 91–100

Pohl K, Böckle G, van der Linden FJ (2005) Software Product Line Engineering: Fundations, Principles and Techniques. Springer–Verlag, Berlin Heidelberg New York

Sinnema M, Deelstra S (2007) Classifying variability modeling techniques. Information & Software Technology 49(7):717–739

Sinnema M, Deelstra S, Nijhuis J, Bosch J (2004) COVAMOF: A framework for modeling variability in software product families. In: Third Software Product Line Conference, Springer –Verlag, LNCS, vol 3154, pp 197–213

Trinidad P, Benavides D, Durn A, Ruiz-Cortés A, Toro M (2008a) Automated error analysis for the agilization of feature modeling. Journal of Systems and Software 81(6):883–896

Trinidad P, Benavides D, Ruiz-Cortés A, Segura S, Jimenez A (2008b) Fama framework. In: 12th Intl. Software Product Line Conference - Tool Demonstrations, IEEE Computer Society, pp 359 –359

Tsang E (1993) Foundations of Constraint Satisfaction. Academic Press, London and San Diego

Tun TT, Boucher Q, Classen A, Hubaux A, Heymans P (2009) Relating requirements and feature configurations: a systematic approach. In: Software Product Lines, 13th International Conference, SPLC 2009, ACM International Conference Proceeding Series, vol 446, pp 201–210

White J, Dougherty B, Schmidt DC (2009) Selecting highly optimal architectural feature sets with filtered cartesian flattening. Journal of Systems and Software 82(8):1268–1284

Appendix A

Table 8 RFW excludes and requires dependencies

Variation Point	Variant	Type	Variation Point	Variant
VP1:Type of vehicle		requires	VP9:Warning signs	
VP1:Type of vehicle		requires	VP10:Signs giving orders	
VP1:Type of vehicle		requires	VP8:Prohibition signs	
VP1:Type of vehicle		requires	VP7:Other signs	
	V10:Hazardous situation alarm	requires		V16:Display and sound indication
	V15:Show on display	excludes		V10:Hazardous situation alarm
	V9:Sound at warning signs	requires		V12:Display and sound indication
	V9:Sound at warning signs	requires		V34:Danger
	V11:Show warning sign	excludes		V9:Sound at warning signs
	V8:Overspeed warning	requires		V48:Maximum speed of x km/h end
	V8:Overspeed warning	requires		V19:City limit
	V8:Overspeed warning	requires		V21:Home zone entry
	V7:No stopping warning	requires		V33:No stopping
	V7:No stopping warning	requires		V13:Warn for no stopping sign
	V7:No stopping warning	excludes		V14:No warning
	V1:Medium-class car	requires		V26:No vehicles
	V1:Medium-class car	requires		V27:No cars
	V1:Medium-class car	requires		V31:Do not enter
	V1:Medium-class car	requires		V41:No overtaking
	V1:Medium-class car	requires		V5:Switchable
	V2:Upper-class car	requires		V26:No vehicles
	V2:Upper-class car	requires		V27:No cars
	V2:Upper-class car	requires		V31:Do not enter
	V2:Upper-class car	requires		V41:No overtaking
	V2:Upper-class car	requires		V6:Durable
	V2:Upper-class car	requires		V8:Overspeed warning
	V3:Small truck (3,5t)	requires		V26:No vehicles
	V3:Small truck (3,5t)	requires		V27:No cars
	V3:Small truck (3,5t)	requires		V31:Do not enter
	V3:Small truck (3,5t)	requires		V41:No overtaking
	V3:Small truck (3,5t)	requires		V5:Switchable
	V4:Big truck (7,5t)	requires		V29:No vehicles w/ weight > 3,5t
	V4:Big truck (7,5t)	requires		V30:No vehicles over max gross weight g > x
	V4:Big truck (7,5t)	requires		V43:No overtaking vehicles > 3,5t
	V4:Big truck (7,5t)	requires		V41:No overtaking
	V4:Big truck (7,5t)	requires		V6:Durable
	V17:Emergency brake	requires		V10:Hazardous situation alarm
	V53:GPS	excludes		V55:Small

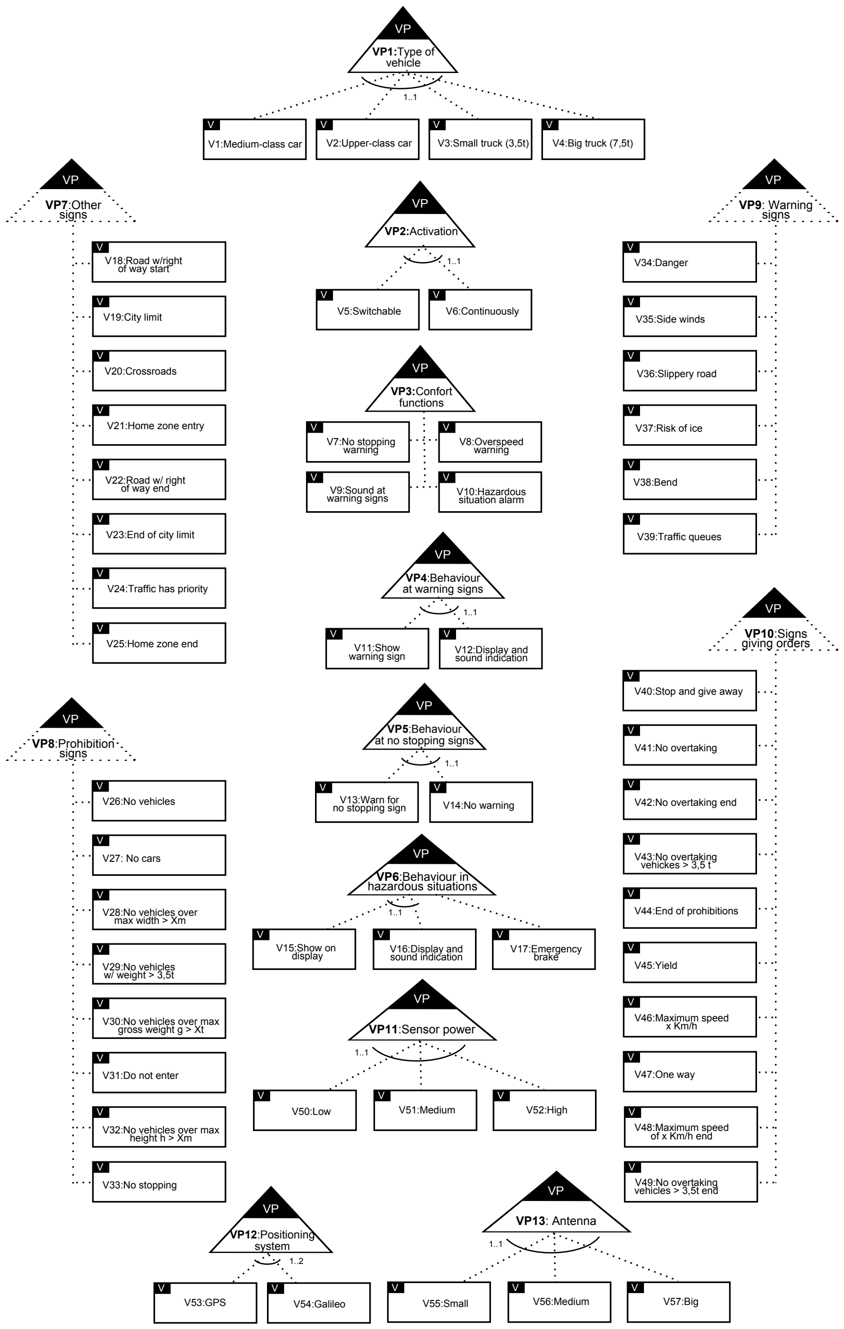


Fig. 19 RFW orthogonal variability model without excludes and requires dependencies

```

V10Hazardous situation alarm          IMPLIES VP11.Range >= 50;
V17Emergency brake                    IMPLIES VP11.Range >= 50;
V18Road w/ right of way start         IMPLIES VP11.Range >= 25;
V19City limit                         IMPLIES VP11.Range >= 50;
V20Crossroads                         IMPLIES VP11.Range >= 25;
V21Home zone entry                    IMPLIES VP11.Range >= 25;
V22Road w/ right of way end           IMPLIES VP11.Range >= 10;
V23End of city limit                  IMPLIES VP11.Range >= 10;
V24Traffic has priority               IMPLIES VP11.Range >= 10;
V25Home zone end                      IMPLIES VP11.Range >= 10;
V26No vehicles                        IMPLIES VP11.Range >= 25;
V27No cars                            IMPLIES VP11.Range >= 25;
V28No vehicles over max width $>$ Xm  IMPLIES VP11.Range >= 25;
V29No vehicles w/ weight $>$ 3.5t     IMPLIES VP11.Range >= 25;
V30No vehicles over max gross weight g $>$ Xt IMPLIES VP11.Range >= 25;
V31Do not enter                       IMPLIES VP11.Range >= 25;
V32No vehicles over max height h $>$ Xm IMPLIES VP11.Range >= 25;
V33No stopping                        IMPLIES VP11.Range >= 10;
V34Danger                             IMPLIES VP11.Range >= 50;
V35Side winds                         IMPLIES VP11.Range >= 50;
V36Slippery road                      IMPLIES VP11.Range >= 50;
V37Risk of ice                        IMPLIES VP11.Range >= 50;
V38Bend                               IMPLIES VP11.Range >= 80;
V39Traffic queues                     IMPLIES VP11.Range >= 80;
V40Stop and give way                  IMPLIES VP11.Range >= 50;
V41No overtaking                      IMPLIES VP11.Range >= 50;
V42No overtaking end                  IMPLIES VP11.Range >= 50;
V43No overtaking vehicles $>$ 3.5t    IMPLIES VP11.Range >= 50;
V44End of prohibitions                IMPLIES VP11.Range >= 25;
V45Yield                              IMPLIES VP11.Range >= 80;
V46Maximum speed X Km/h               IMPLIES VP11.Range >= 50;
V47One way                            IMPLIES VP11.Range >= 25;
V48Maximum speed of X Km/h end        IMPLIES VP11.Range >= 10;
V49No overtaking vehicles $>$3.5t end IMPLIES VP11.Range >= 10;
V11Show warning sign                  IMPLIES TotalAccuracy <= 30;
V12Display and sound indication       IMPLIES TotalAccuracy <= 30;
V13Warn for no stopping sign          IMPLIES TotalAccuracy <= 10;
V15Show on display                    IMPLIES TotalAccuracy <= 30;
V16Display and sound indication       IMPLIES TotalAccuracy <= 30;
V17Emergency brake                    IMPLIES TotalAccuracy <= 10;
V18Road w/ right of way start         IMPLIES TotalAccuracy <= 30;
V19City limit                         IMPLIES TotalAccuracy <= 30;
V20Crossroads                         IMPLIES TotalAccuracy <= 30;
V21Home zone entry                    IMPLIES TotalAccuracy <= 30;
V22Road w/ right of way end           IMPLIES TotalAccuracy <= 30;
V23End of city limit                  IMPLIES TotalAccuracy <= 30;
V24Traffic has priority               IMPLIES TotalAccuracy <= 30;
V25Home zone end                      IMPLIES TotalAccuracy <= 30;
V26No vehicles                        IMPLIES TotalAccuracy <= 10;
V27No cars                            IMPLIES TotalAccuracy <= 10;
V28No vehicles over max width $>$ Xm  IMPLIES TotalAccuracy <= 10;
V29No vehicles w/ weight $>$ 3.5t     IMPLIES TotalAccuracy <= 10;
V30No vehicles over max gross weight g $>$ Xt IMPLIES TotalAccuracy <= 10;
V31Do not enter                       IMPLIES TotalAccuracy <= 10;
V32No vehicles over max height h $>$ Xm IMPLIES TotalAccuracy <= 10;
V33No stopping                        IMPLIES TotalAccuracy <= 10;
V34Danger                             IMPLIES TotalAccuracy <= 30;
V35Side winds                         IMPLIES TotalAccuracy <= 30;
V36Slippery road                      IMPLIES TotalAccuracy <= 30;
V37Risk of ice                        IMPLIES TotalAccuracy <= 30;
V38Bend                               IMPLIES TotalAccuracy <= 10;
V39Traffic queues                     IMPLIES TotalAccuracy <= 30;
V40Stop and give way                  IMPLIES TotalAccuracy <= 10;
V41No overtaking                      IMPLIES TotalAccuracy <= 30;
V42No overtaking end                  IMPLIES TotalAccuracy <= 30;
V43No overtaking vehicles $>$ 3.5t    IMPLIES TotalAccuracy <= 30;
V44End of prohibitions                IMPLIES TotalAccuracy <= 30;
V45Yield                              IMPLIES TotalAccuracy <= 10;
V46Maximum speed X Km/h               IMPLIES TotalAccuracy <= 30;
V48Maximum speed of X Km/h end        IMPLIES TotalAccuracy <= 30;
V49No overtaking vehicles $>$3.5t     IMPLIES TotalAccuracy <= 30;
```

Fig. 20 RFW domain constraints