

# Separating Concerns in Feature Models

## Retrospective and Support for Multi-Views

Arnaud Hubaux, Mathieu Acher, Thein Than Tun, Patrick Heymans,  
Philippe Collet and Philippe Lahire

**Abstract** Feature models (FMs) are a popular formalism to describe the commonality and variability of a set of assets in a software product line (SPL). SPLs usually involve large and complex FMs that describe thousands of features whose legal combinations are governed by many and often complex rules. The size and complexity of these models is partly explained by the large number of concerns considered by SPL practitioners when managing and configuring FMs. In this chapter, we first survey concerns and their separation in FMs, highlighting the need for more modular and scalable techniques. We then revisit the concept of view as a simplified representation of an FM. We finally describe a set of techniques to specify, visualize and verify the coverage of a set of views. These techniques are implemented in complementary tools providing practical support for feature-based configuration and large scale management of FMs.

---

Arnaud Hubaux, Patrick Heymans  
PReCISE Research Centre, University of Namur, Belgium  
e-mail: `ahu,phe@info.fundp.ac.be`

Mathieu Acher  
University of Rennes 1, Irisa and INRIA, France  
PReCISE Research Centre, University of Namur, Belgium  
e-mail: `mathieu.acher@irisa.fr`

Thein Than Tun  
Department of Computing, The Open University, UK  
e-mail: `t.t.tun@open.ac.uk`

Philippe Collet, Philippe Lahire  
Université de Nice Sophia Antipolis - I3S (CNRS UMR 6070), France  
e-mail: `collet,lahire@i3s.unice.fr`

## 1 Introduction

In many application domains, such as avionics, telecommunications or automotive, organizations build software-intensive systems that are similar to each other. Rather than re-developing each system from scratch, these organizations reuse common software artifacts on a large scale.

The paradigm of *software product line* (SPL) engineering has emerged to support the modeling and development of software system families rather than individual systems. It aims at efficiently producing and maintaining multiple similar software products. This is analogous to the automotive industry, where the focus is on creating a single production line, out of which many customized but similar variations of a car model are produced. The key principle is to institutionalise reuse throughout the development process to obtain economies of scale and scope [53]. To achieve reuse, SPL engineering is usually separated in two complementary phases: *domain engineering* and *application engineering*. Domain engineering starts with *domain analysis*, which documents commonality (i.e., common parts of products) and variability (i.e., differences between products). Reusable assets that satisfy these descriptions are then modelled and implemented. During application engineering, the required assets are selected and possibly extended to derive, as quickly and efficiently as possible, an appropriate product. To be successful, the investments required to develop the reusable artifacts during domain engineering must be outweighed by the benefits of deriving the individual products during application engineering [25]. Domain analysis is therefore a crucial phase.

To date, feature modeling has been recognized as one of the most popular domain analysis techniques. Introduced in the 1990's and now widely adopted, *feature models* (FMs) are a simple formalism whose main purpose is to document variability in terms of *features*, i.e., domain abstractions or functionalities relevant to stakeholders [19]. The main concepts of the language are features and relationships between features. FMs have been given a formal semantics [59] which opened the way for safe and efficient automation of various, otherwise error-prone and tedious tasks such as consistency checking, FM merging and product counting. A repertoire of such automations can be found in [12].

A particular type of automation is *feature-based configuration* (FBC). FBC is an interactive process during which one or more stakeholders select and discard features to build a specific product. Traditionally, FBC systems support FM modelling, analysis and configuration. Currently, FBC techniques and tools facilitate the work of stakeholders in various ways, including: decision verification and propagation [47, 38, 22]; auto-completion [21, 38]; scheduling of configuration tasks [23, 20, 35]; and alternative representations of FMs [15, 17].

FMs, and therefore FBC, are becoming increasingly large and complex. FMs are not only used to describe variability in software designs, but also

variability in different contexts, at different times in the development, and in different parts of the system [41, 57, 50, 32, 20]. Consequently, the list of *concerns* that may be considered in an FM is very comprehensive [64, 9, 34] ranging from hardware description [41], organizational structure [57], business to implementation details [50]. Concerns are related in numerous ways and there can be thousands of features whose legal combinations are governed by many and often complex rules.

Furthermore, it has been observed that maintaining a single large FM for the entire system may not be feasible [54, 26]. With FMs being increasingly complex, describing various concerns of an SPL and handled by several stakeholders (or even different organizations), managing them with a large number of related features is intuitively a problem of *separation of concerns* (SoC) [61, 11]. The sought benefits are indeed similar to the ones of software engineering disciplines, i.e., reduced complexity, improved reusability and simpler evolution [61]. A possible way to achieve SoC is then to rely on views, i.e., simplified representations of an FM tailored for a specific stakeholder, role, or task [36]. Views facilitate the decision-making process in that they only focus on those parts of the FM that are relevant for a given concern.

In this chapter, our goal is to give a clear overview of existing approaches in the field and state-of-the-art techniques for separating concerns in FMs. The intended audience is domain analysts or SPL practitioners working with FMs with an interest for FBC. In the first part of this chapter, we present a review of SoC in FMs. SoC has spawned much research on FM separation, composition and analysis. Here, we reuse some material presented in [36] and focus on concerns and their *separation* in FMs and FBC. We highlight the need for more modular and scalable techniques and revisit the concept of views. In the second part of this chapter, we focus on the creation of consistent views and the generation of alternative visualisations for FBC. We present and compare two techniques to synthesize visualisations of an FM. We also report on the progress made in developing tool support for SoC and multi-view FBC.

The rest of this chapter is organised as follows. Section 2 re-examines the basics of FMs and introduces our working example. Section 3 presents the general problem of SoC in FM and reviews existing works in the field. Section 4 describes a set of SoC techniques to specify, automatically generate, and check multiple views. Section 5 presents the tools supporting it.

## 2 Background






### 2.1 Feature-based Configuration

Schobbens *et al.* [59] defined a generic formal semantics for a wide range of FM dialects. In essence, an FM  $d$  is a hierarchy of features (typically a tree) topped by a root feature. An FM is informally defined as follows.

**Definition 1 (FM (adapted from [59])).** *An FM  $d$  is a tuple  $(N, r, \lambda, DE, \Phi)$  where  $N$  denotes the set of features.  $r \in N$  the root of the feature tree.  $\lambda : N \rightarrow \mathbb{N} \times \mathbb{N}$  denotes the cardinality  $\langle i..j \rangle$  attached to a feature, where  $i$  (resp.  $j$ ) is the minimum (resp. maximum) number of children (i.e. features at the level below) required in a product (aka configuration). For convenience, common cardinalities are denoted by Boolean operators, as shown in Table 1.  $DE \subseteq N \times N$  denotes the decomposition edges, i.e., the parent-child relationship. Additional constraints that crosscut the tree ( $\Phi \in \mathbb{B}(N)$ ) can also be added and are defined, without loss of generality, as a conjunction of Boolean formulae.*

The semantics of an FM is the set of configurations (also called products), denoted  $\llbracket d \rrbracket$ , where each configuration is a combination of selected features. The full syntax and semantics as well as benefits, limitations and applications of FMs are extensively discussed elsewhere [59, 12].

**Table 1** FM decomposition operators

Concrete syntax					
Decomposition operator	and: $\wedge$	or: $\vee$	xor: $\oplus$	generalized cardinality	optional
Cardinality	$\langle n..n \rangle$	$\langle 1..n \rangle$	$\langle 1..1 \rangle$	$\langle i..j \rangle$	$\langle 0..1 \rangle$

FBC tools use FMs to pilot the configuration of customisable products. These tools usually render FMs in an *explorer-view* style [55, 47], as shown in the upper part of Table 1. The tick boxes in front of features are used to capture decisions, i.e. whether the features are selected or not. We now illustrate the FM abstract syntax more concretely on our working example.

## 2.2 Working Example

*Audi* is a German car manufacturer. Nowadays, Audi offers 12 different model lines, each available in different body styles, each broken down in different models. This paper will focus on the *Audi A3*, with the *sportback* body style. An example of its configurator in action is shown in Fig. 1. The two FMs in Fig. 2 are samples reverse engineered from the car configurator<sup>1</sup> for the A3 and RS3 models.

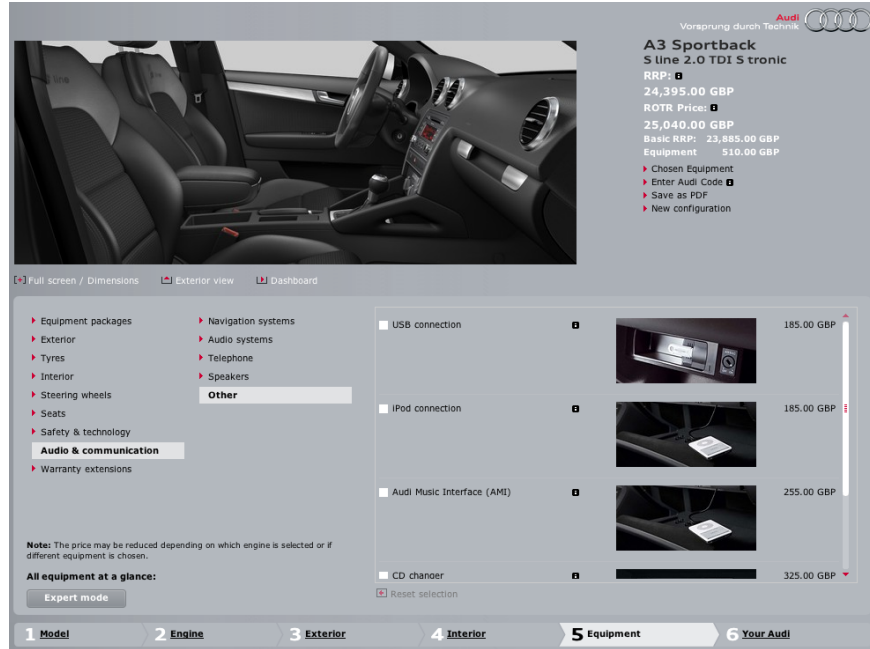
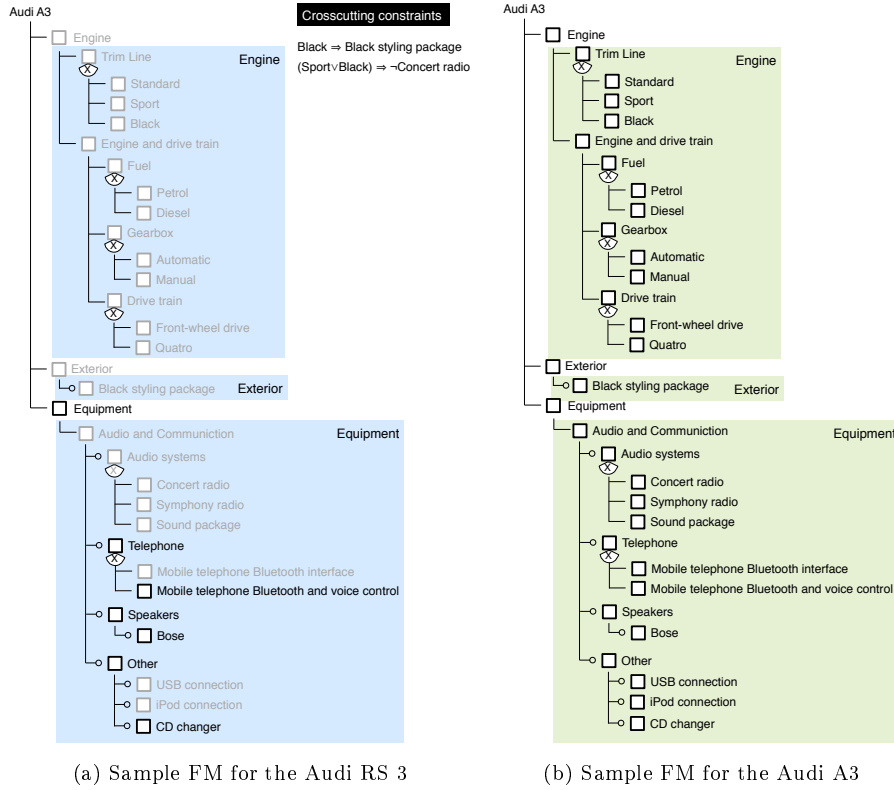


Fig. 1 Screenshot of Audi A3 configurator.

Although similar, these models show very different options to customers. The features hidden by the configurator appear in light grey. This, however, does not indicate that the value of these features is not set. It rather means that customers cannot *manually* set their values. In Fig. 2a for instance, none of the *Engine* features are available. Yet, the RS3 has a *Quattro* drive train and a *Petrol* engine. This practice resembles the inactivation of features in operating systems configurators such as those used for Linux and eCos [13]. The isolation of visible from hidden features is thus a first possible criterion to separate concerns.

<sup>1</sup> Reverse engineered from <http://configurator.audi.co.uk/> on January 20th, 2012. Some labels were shortened for conciseness.



**Fig. 2** Two FMs of Audi A3 model line

The second criterion is determined by the steps in the configuration process. As Fig. 1 shows, the configuration follows a number of steps starting from 1. *Model*, going through 5. *Equipment*, and ending at 6. *Your Audi*. This decomposition is illustrated in Fig. 2 by the coloured areas. In contrast to the first criteria, these views are used to progressively disclose the options.

The Audi configurator, like many others (e.g. Linux and eCos [66]), rely on *ad hoc* solutions that usually do not come with a proof of completeness and correctness, and can hardly be reused from one domain to the other. A general and formal foundation for separation of concerns in FBC is necessary. This paper proposes a retrospective on this SoC in FBC and discusses the complementary combination of views and slices to achieve flexible and reliable SoC. Without delving into formal developments, it provides a frame of reference to specify, verify and visualize concurrent concerns on an FM.

### 3 Concerns and Their Separation: Retrospective

A major limitation of current FM languages is that they are found not to scale well when applied to realistic SPLs. In real projects it has been reported that maintaining a single large FM for the entire system may not be feasible [26].

Firstly, FMs are increasingly larger with possibly thousands of features related by numerous complex constraints. As an extreme case, the variability model of Linux exhibits 6000+ features [60]. Secondly, various concerns of an SPL, handled by several stakeholders (or even different organizations), should be properly modeled and managed. As a result, FMs quickly become too complex to be understood and managed by practitioners. In FBC context, it is very hard for a practitioner to consider thousands of configuration options as a whole.

The principle of SoC points to an effective way to manage the size and complexity of FMs. On the one hand, several FMs may be originally separated and combined, for instance, when engineers describe the variability of modular systems (e.g., software components or services [5]), when independent suppliers describe the variability of their different products in software supply chains [22, 33], or when a multiplicity of SPLs must be combined [32, 26]. On the other hand, it may be the intention of an SPL practitioner to modularize the variability description of the system according to different criteria or concerns such as external vs. internal variability [54, 50, 7], abstraction layers [41], or views tailored for a specific stakeholder [36].

The problem of SoC in FMs has been extensively reported in the literature but there is no consensus on how best to separate and compose these concerns. In this paper, we focus on the *separation of concern* problem. This section highlights key achievements in this domain with a particular emphasis on *views*, which have been repeatedly advocated as a means to solve scalability and configuration issues.

#### 3.1 Variability modelling

Dealing with real-world problems implies dealing with multiple stakeholders with different and often inconsistent perspectives. Viewpoint-based approaches have been around for nearly two decades and address exactly those issues. They mainly support the identification, structuring, reconciliation and co-evolution of heterogeneous requirements [28, 51]. They have been studied mostly by the requirements engineering (RE) community. They are more concerned with the identification and reconciliation of viewpoints than with the specification and generation of viewpoint- (or concern-) specific views on an artifact like the FM in our case. Viewpoint-based RE techniques are not specific to SPLE. Still, viewpoint-based techniques can be used *upstream* of variability modelling to help build a consistent FM from heterogeneous view-

points. More specific to variability modelling, Grünbacher *et al.* [31] outline the challenges that arise when heterogeneous stakeholders are involved in the modelling of large FMs.

The identification of stakeholders is also a problem studied in RE [29]. We refer the reader to [30] for a general introduction to stakeholder identification and ways to structure and trace their contributions. Directly related to feature modelling, Bidian *et al.* [14] identify stakeholder profiles through the tasks appearing in goal models which are subsequently linked to the features realising them.

### 3.2 View specification

Early attempts to manage the complexity of FMs [40, 41] were mainly concerned with separating user-oriented from technical features. For this, simple techniques were used, namely annotation and layering of the FM, but those remained informal and were not used to generate views or for configuration. In OVM [53], a similar distinction was proposed between internal and external variability, but had the same limitations as the aforementioned approaches.

Zhao *et al.* [67] group features according to stakeholder profiles and other typical concerns. A major limitation is that they do not display decomposition operators in views, which greatly simplifies the problem at the expense of completeness. Features in views are physically duplicated and mapped to features of the FM. The resulting links are represented as constraints between the views and the FM.

Researchers developed SoC techniques for FMs that reflect organisational structures and tasks. Reiser *et al.* [56] address the problem of representing and managing FMs in SPLs that are developed by several companies, as is common for example in the automotive industry. They propose to use several FMs and structure them hierarchically. This way, each of them can be managed separately by one of the partner companies. Local changes are then propagated to other FMs through the hierarchy. Hierarchical decomposition in SPLs was also studied by Thompson *et al.* [62], although not in relation to FMs.

Clarke *et al.* [18] introduce a formal theory of views for FMs, where a view is defined as a disjoint set of features and *abstractions*. An abstraction encapsulates a set of features hidden behind a label meaningful to the user. They formally define compatibility properties between views and their reconciliation, i.e. combination. To preserve the genericity of their mathematical model, the authors reason exclusively in terms of features independently of the structure and constraints imposed by the FM. As a result, they do not discuss the concrete specification, rendering, and configuration of views on an FM.



### 3.3 Configuration

Reiser *et al.* [56] along with Mannion *et al.* [44] discuss how multiple views affect the structure of the FM and configuration with a particular focus on decision propagation and conflict resolution [44]. Unlike other approaches that only consider the selection/deselection of features, they address changes to the structure of views that are propagated back to the original FM. To resolve conflicts that can happen during the merge of concurrent changes, they propose a list of conflict resolution rules within views. They thus focus on resolving conflicts among changes to the content of the FM rather than conflicts between configuration decisions.

Batory *et al.* [10] have worked on multi-dimensional SoC where a dimension is a set of features addressing a particular concern. They use a so-called “origami matrix” to describe the relationships between features across the dimensions. Their approach does not aim to generate views but rather to compose features (described separately) along each dimension.

Czarnecki *et al.* [23] have introduced multi-level staged configuration as a way of organizing FBC as a sequence of stages. This idea was later formalised [20] and extended [35] to deal with arbitrarily complex configuration processes (not only purely sequential ones). Mendonça *et al.* [46, 48] suggest configuration spaces (similar to views) as a means to support collaborative product configuration. They also provide algorithms to automatically generate a configuration plan out of an FM and a set of configuration spaces. Although these and related [50, 63] approaches are automatable and readily applicable to configuration, they remain limited to a single “tyrannical” decomposition scheme [61] (e.g., stages or workflow activities) which must be decided in advance.

Over the years, various interactive FBC environments have been developed (e.g. [8, 55, 45, 42]). Based on formal semantics, these tools use solvers (e.g. SAT, BDD and CSP) to propagate decisions throughout the FM and ensure the global consistency of the final product. Commercial FBC tools (e.g. [55, 42]) also offer integration with popular modelling environments like IBM Rational or Simulink. Traditionally, FBC tools assume that there exists a single monolithic FM and do not account for configuration processes that are distributed among various stakeholders who have specific concerns and who intervene at different moments [48, 46]. Without the appropriate support, FBC can become very cumbersome and error-prone, e.g., if a single stakeholder has to make decisions on behalf of all others [48].

## 4 Separating Concerns in Feature Models

### 4.1 Views

#### 4.1.1 Basic definition

Separating concerns requires the ability to specify the parts of the FM that are of interest and the person(s) who can configure it. In order to achieve this, the FM can be augmented with a set  $V$  of views, each of which consists of a set of features. Formally, a *multi-view FM* is defined as follows:

**Definition 2 (Multi-view FM [36]).** *A multi-view FM  $m$  is a tuple  $(N, r, \lambda, DE, \Phi, V)$  where  $V = \{v_1, v_2, \dots, v_n\}$  is the multiset of views such that:*

- $N, r, \lambda, DE, \Phi$  conform to Definition 1;
- $\forall v_i \in V \bullet v_i \subseteq N \wedge r \in v_i$ .

Therefore, for any concern that requires only partial knowledge of the FM, such as a profile, a view can be defined. We also consider that the root is part of each view.  $V$  is a multiset to account for duplicated sets of features.

#### 4.1.2 View specification

We distinguish between two ways of specifying views. With *extensional definitions*, the features that appear in a view are enumerated, or tagged so as to indicate the view to which each of the features belongs. A drawback is that the process of enumerating and tagging can be time-consuming and error-prone without appropriate tool support.

With *intensional definitions*, the features in a view are defined according to a query defined on the FM. For instance, the tree structure of the FM can be exploited by languages like XPath to specify the views. A major drawback of intensional definitions is that textual languages may not be as intuitive as graphical approaches for casual users. Furthermore, it is harder to maintain consistency between the FM and the textual expressions when the diagram evolves without proper tool support.

Having said that, extensional and intensional definitions can be used together in practice. Textual expressions corresponding to intensional specifications could be generated from a graphical view definition tool. Conversely, it is possible to generate feature tags from textual expressions and link them to the features in the expression. These links can then be used to trace changes from the FM back in the expression. This allows us to overcome the limitations of both extensional and intensional definitions. In the following discussions, we refer to features contained in views irrespective of the specification method.

### 4.1.3 View coverage

An important property that should be guaranteed by an FBC system is that all configuration questions are eventually answered [20]. In a multi-view context, one may consider enforcing the following condition.

**Definition 3 (Sufficient coverage condition [36]).** *For a view  $v$  of a multi-view FM  $m$  the sufficient coverage condition is:*

$$\bigcup_{v \in V} v = N$$

Intuitively, this means that all the features appear in at least one view, hence no feature can be left undecided.<sup>2</sup> Although sufficient, this is not a necessary condition because some decisions can usually be deduced from others.

A *necessary condition* can be defined in terms of propositional definability [43]. It is necessary to ensure that the decisions on the features that do not appear in any view can be inferred from (i.e. are *propositionally defined by*) the decisions made on the features that are part of the view. In the following definition,  $\text{defines}(F, f)$  denotes the propositional definability of  $f$  by  $F$ .

**Definition 4 (Necessary coverage condition [36]).** *For a view  $v$  of a multi-view FM  $m$  the necessary coverage condition is:*

$$\forall f \notin \bigcup_{v \in V} v \bullet \text{defines}\left(\bigcup_{v \in V} v, f\right)$$

$\text{defines}$  can be evaluated by translating the FM into an equivalent propositional formula (done in linear time [58]) and by applying the SAT-based algorithm described in [43]. Although this check is NP complete in theory, it is not expected to be a problem in practice, since SAT solvers can handle FMs with thousands of features.

Features in  $N \setminus \bigcup_{v \in V} v$  that do not satisfy the above condition will have to be integrated in existing views, or extra constraints will have to be added to determine their value.

In application domains such as operating systems, features such as those used for calculating the boot entry to use are hidden from users [13], and may not be visible in any view. In such cases, the verification of the necessary condition determines whether the value of the hidden features can be derived from the features in the views.

However, in cases such as the Audi configurator (Section 2.2), these two conditions are too strict. Assuming that a view only contains the features relevant to a customer, it will naturally not contain the hidden features. In this particular case, some hidden features might not be decided upon. Some

---

<sup>2</sup> Note that the complete view coverage is usually assumed by multi-view approaches (e.g. [48]).

existing configurators, such as the one of Linux, nullify these features. In this context, the necessary coverage condition has to be adapted such that one only checks features that are neither in  $\bigcup_{v \in V} v$  nor in the nullified features.

## 4.2 Visualisation

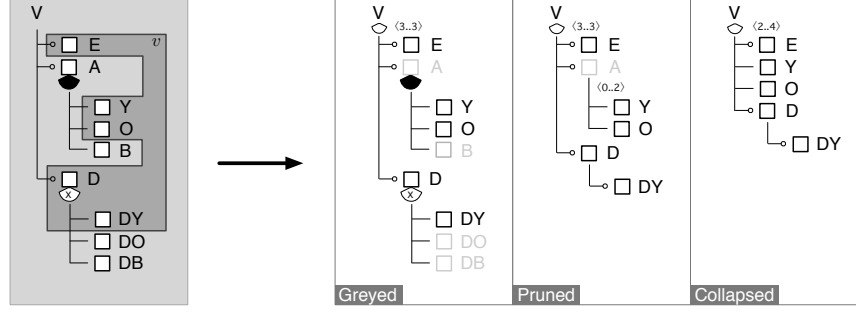
Although views are abstract, they have to be made concrete to be used during FBC. A concrete view is called a *visualisation*. A visualisation strives to find a compromise between not showing in a view features that do not belong to the view, and showing features that do not belong to the view but indirectly provide context for features that should be shown. In Fig. 4.2, for instance, feature *Y* is in the view (darker area), but its parent feature *A* is not.

To tackle this problem of view rendering, we have observed the practice of developers (see [37] for more details about the case study and our experience with PloneMeeting) and discussed with them alternative visualisations. Our discussions included the relative merits of the approaches suggested in [67, 48], and the filtering mechanisms provided by tools such as `pure::variants` [55], and kernel configurators for operating systems (e.g. `xconfig` for Linux and `configtool` for eCos [27]). These tools provide simple filtering or search mechanisms that are similar to views on an FM. In these cases, a filter is a regular expression on the FM. Any feature matching the regular expression is displayed typically without any control on the location of the feature in the hierarchy. Interestingly, all these approaches produce purely graphical modifications (e.g. by greying out irrelevant features) whereas cardinalities are not recomputed.

The main outcome of our investigation is a set of four complementary visualisations offering different levels of details, as depicted in Fig. 3. The darker area defines a specific view of the FM, called *v*. These views were built to present information on a *need-to-know* basis. The amount of information displayed can be regulated, while providing enhanced control over access rights. For instance, there is always a standardised configuration menu that can display the position of the feature in the hierarchy and hide unavailable options. On the other hand, in a critical application, features outside a view may have to be protected as trade secrets. Therefore, visualisations not only can provide convenient representations of a view, but they can also restrict the information a stakeholder can access.

Fig. 3 illustrates the alternative visualisations of FM views we propose:

- The *greyed* visualisation is a mere copy of the whole FM except that the features that do not belong to the view are greyed out (e.g. *A*, *B*, *DO* and *DB*). Greyed features are only displayed but cannot be manually selected/deselected.



**Fig. 3** Three alternative visualisations of FM views: greyed, pruned and slice/collapsed [36].

- In the *pruned* visualisation, features that are not in the view are pruned (e.g. *B*, *DO* and *DB*) unless they appear on a path between a feature in the view and the root, in which case they are greyed out (e.g. *A*).
- In the *collapsed* visualisation, all the features that do not belong to the view are pruned. A feature in the view whose parent or ancestors are pruned is connected to the closest ancestor that is still in the view. If no ancestor is in the view, the feature is directly connected to the root (e.g. *Y* and *O*).
- The *slice* visualisation is similar to the *collapsed* visualisation except that it takes cross-tree constraints into consideration. Consequently, decomposition operators might be altered to preserve the correctness of these constraints.

Generating visualisations, from an FM and a view, is a form of FM transformation.

**Definition 5 (View visualisation).** *The visualisation of a view  $v$  is the transformation of the original FM into a new FM  $d_v^t = (N_v^t, r, \lambda_v^t, DE_v^t, \Phi)$ , where  $t$ , the type of visualisation, can take one of four values:  $g$  (greyed),  $p$  (pruned),  $c$  (collapsed), and  $s$  (slice).*

The greyed visualisation is the simplest case because there is no transformation beyond the greying of each feature  $f \notin v$  (i.e.  $d_v^g = d$ ). The transformations for the pruned and collapsed visualisations, on the other hand, filter nodes, remove dangling decomposition edges and adapt the cardinalities accordingly.

#### 4.2.1 Pruned visualisation

$N_v^p$ , the set of features in this visualisation, is the subset of  $N$  limited to features that are in  $v$  or have a descendant in  $v$ . The definition uses  $DE^+$ ,

the transitive closure of  $DE$ . Based on  $N_v^p$ , we remove all dangling edges, i.e. those not in  $N_v^p \times N_v^p$  to create  $DE_v^p$ .

**Transformation 1 (Pruned visualisation [36])** *The transformations applied to the FM to generate the pruned visualisation are:*

$$\begin{aligned} N_v^p &= \{n \in N \mid n \in v \vee \exists f \in v \bullet (n, f) \in DE^+\} \\ DE_v^p &= \{DE \cap (N_v^p \times N_v^p)\} \\ \lambda_v^p(f) &= (\text{mincard}_v^p(f), \text{maxcard}_v^p(f)) \end{aligned}$$

In order to compute the new cardinalities  $\lambda_v^p(f)$ ,  $\text{mincard}_v^p(f)$  and  $\text{maxcard}_v^p(f)$  are defined as follows:

$$\begin{aligned} \text{mincard}_v^p(f) &= \max(0, \lambda(f).min - |\text{orphans}_v^p(f)|) \\ \text{maxcard}_v^p(f) &= \min(\lambda(f).max, |\text{children}(f)| - |\text{orphans}_v^p(f)|) \end{aligned}$$

where  $\text{orphans}_v^p(f) = \text{children}(f) \setminus N_v^p$  i.e., the set of children of  $f$  that are not in  $N_v^p$ .  $\lambda(f).min$  and  $\lambda(f).max$  represent the minimum and maximum values of the original cardinality, respectively. For the minimum, the difference between the cardinality and the number of orphans can be negative in some cases, hence the necessity to take the maximum between this value and 0. The maximum value is the maximum cardinality of  $f$  in  $d$  if the number of children in  $v$  is greater. If not, the maximum cardinality is set to the number of children that are in  $v$ .

#### 4.2.2 Collapsed visualisation

The set of features  $N_v^c$  of this visualisation is simply the set of features in  $v$ . The consequence on  $DE_v^c$  is that some features have to be connected to their closest ancestor if their parent is not part of the view.

**Transformation 2 (Collapsed visualisation [36])** *The transformations applied to the FM to generate the collapsed visualisation are:*

$$\begin{aligned} N_v^c &= v \\ DE_v^c &= \{(f, g) \mid f, g \in v \wedge (f, g) \in DE^+ \wedge \nexists f' \in v \bullet ((f, f') \in DE^+ \wedge (f', g) \in DE^+)\} \\ \lambda_v^c(f) &= (\text{mincard}_v^c(f), \text{maxcard}_v^c(f)) \end{aligned}$$

The computation of cardinalities  $\lambda_v^c(f)$  is slightly more complicated than in the pruned case. Formally,  $\text{mincard}_v^c(f)$  and  $\text{maxcard}_v^c(f)$  are defined as follows:

$$\begin{aligned} \text{mincard}_v^c(f) &= \sum \text{min}_{\lambda(f).min}(\text{ms\_min}_v^c(f)) \\ \text{maxcard}_v^c(f) &= \sum \text{max}_{\lambda(f).max}(\text{ms\_max}_v^c(f)) \end{aligned}$$

where

$$\begin{aligned} \text{ms\_min}_v^c(f) &= \{\text{mincard}_v^c(g) \mid g \in \text{orphans}_v^c(f)\} \uplus \{1 \mid g \in \text{children}(f) \setminus \text{orphans}_v^c(f)\} \\ \text{ms\_max}_v^c(f) &= \{\text{maxcard}_v^c(g) \mid g \in \text{orphans}_v^c(f)\} \uplus \{1 \mid g \in \text{children}(f) \setminus \text{orphans}_v^c(f)\} \end{aligned}$$

The multisets  $\text{ms\_min}_v^c(f)$  and  $\text{ms\_max}_v^c(f)$  collect the cardinalities of the descendants of  $f$ . The left part of the union<sup>3</sup> recursively collects the

<sup>3</sup>  $\uplus$  is the union on multisets.

cardinalities of the collapsed descendants whereas the right side adds 1 for each child that is in the view. The  $\lambda(f).min$  minimum values of the multiset are then summed to obtain the minimum cardinality of  $f$ . The maximum value is computed similarly.

### 4.2.3 Slice visualisation

We revisit here a technique called *slicing* that, given an FM (typically large), produces a new, smaller FM containing only a subset of features of the input FM. We show that the slicing can be used to synthesize visualisations.

The overall idea behind FM slicing is similar to program slicing [65]. Program slicing has been successfully applied in computer programming and aims at simplifying or abstracting programs by focusing on selected aspects of semantics. Program slicing techniques usually proceed in two steps: the subset of elements of interest (e.g. a set of variables of interest and a program location), called the slicing *criterion*, is first identified ; then, a *slice* (e.g. a subset of the source code) is computed. In the context of FMs, we define the slicing criterion as a set of features considered to be pertinent by an SPL practitioner while the slice is a new FM (see Transformation 3).

*Slicing semantics.* The major preoccupation for an SPL practitioner is the legal combination of features (configurations) defined by an FM. The same observation applies when decomposing the FM into smaller concerns. We want to guarantee semantic properties of smaller parts, i.e., in terms of set of configurations. Nevertheless, several FMs, yet with different hierarchies, can represent a given set of configurations. Therefore, the semantics of the slicing operator is defined both in terms of set of configurations and feature hierarchy (see Transformation 3).

**Transformation 3 (Slice visualisation [4])** *We define slicing as an operation on FM, denoted  $\Pi_{N_v^s}(d) = d_v^s$  where  $N_v^s$  is a set of features, called the slicing criterion, and  $d_v^s$  is a new FM, called the slice.*

*The result of the slicing operation is a new FM,  $d_v^s$ , such that:*

- **Feature hierarchy:** *Features of the hierarchy include the slicing criterion of the original FM while features are connected to their closest ancestor if their parent feature is not part of the slice FM. It corresponds to the feature hierarchy defined for the collapsed visualisation (see Transformation 2).*
- **Configuration semantics:** *The valid configurations,  $\llbracket d_v^s \rrbracket$ , one could infer from a slice are actually the valid configurations of the original FM, when looking only at the slicing criterion features  $N_v^s$ . Formally, the projected<sup>4</sup> set of configurations is defined as  $\llbracket d_v^s \rrbracket = \llbracket d \rrbracket|_{N_v^s}$ .*

<sup>4</sup> For two given sets  $A$  and  $B$ , we note  $A|_B$  the projection of  $A$  on  $B$  such that:

$$A|_B \triangleq \{a' \mid a \in A \wedge a' = a \cap B\} = \{a \cap B \mid a \in A\}$$

It should be noted that the hierarchy of the slice FM corresponds to the hierarchy defined for the collapsed visualisation (see the right hand side of Fig. 3). In the following, we will describe an algorithm to synthesize automatically such visualisations.

*Automated slice synthesis.* Our previous experience has shown that *syntactic* strategies have severe limitations to accurately represent a given set of configurations (as expected by Transformation 3), especially in the presence of cross-tree constraints [2]. The same observation applies for the slicing operation so that we reason directly at the *semantic* level. The key idea of the proposed algorithm is to (i) compute the propositional formula representing the projected set of configurations, and then to (ii) apply satisfiability techniques to construct a complete FM (including variability information and cross-tree constraints) using the formula. A major difference with previous works [24, 60] that propose to synthesize FMs from propositional formulae is that the feature hierarchy of the resulting FM can be determined and computed (see Transformation 3).

*Formula Computation.* Let  $d_v^s = \Pi_{N_v^s}(d)$ . The propositional formula  $\phi_s$  corresponding to  $d_v^s$  can be defined as follows:

$$\phi_s \equiv \exists f_1, f_2, \dots, f_{m'} \phi$$

where  $f_1, f_2, \dots, f_{m'} \in (N \setminus N_v^s) = N_{removed}$  and  $\phi$  is the encoding of  $d$  as a propositional formula. The propositional formula  $\phi_s$  is obtained from  $\phi$  by *existentially quantifying* out variables in  $N_{removed}$ . Intuitively, all occurrences of features that are not present in any configuration of  $d_v^s$  are removed by existential quantification<sup>5</sup> in  $\phi$ .

*From formula to FM.* From the propositional formula  $\phi_s$ , several FMs can be synthesized [60]. In our case, though, we already *know* what the resulting hierarchy is. Our algorithm exploits this information. We first compute the hierarchy, we then set the variability information (mandatory/optional, Xor and Or-groups) and finally the constraints (bi)-implies/excludes/others.

*Mandatory and feature groups.* At this step, all features, except root, are considered optional. We compute the binary implication graph, noted  $BIG_s$ , of the formula  $\phi_s$  over  $N_v^s$ .

$BIG_s$  is a directed graph  $G = (V, E)$  formally defined as:

$$V = N_s \quad E = \{(f_i, f_j) \mid \phi_s \wedge f_i \Rightarrow f_j\}$$

We use  $BIG_s$  to identify biimplications and thus set mandatory features together with their parents. For feature groups, we reuse the prime implications method proposed in [24], so that we can identify Or- and Xor-groups. An important issue is that a feature may be candidate to several feature groups (which is not allowed by FMs). Therefore some feature groups are

<sup>5</sup> Existential quantification is defined as the substitution of a Boolean variable  $ft$  to True and False values. Formally:  $\exists ft \phi =_{def} \phi|_{ft} \vee \phi|_{\bar{ft}}$  where  $\phi|_{ft}$  (resp.  $\phi|_{\bar{ft}}$ ) denotes the assignment of  $ft$  to True (resp. False) value in  $\phi$ .



dismissed so that FMs are well-formed. We use the original FM to retrieve initial feature groups (see details in [1]).

*Constraints.* The set of implies constraints can be deduced by removing edges of  $BIG_s$  that are already expressed (e.g. parent-child relations). For the purpose of conciseness, some implies constraints can be transformed into equivalence relations (e.g.,  $A \Rightarrow B \wedge B \Rightarrow A$  can be transformed into  $A \Leftrightarrow B$ ). Similarly, excludes constraints are produced by computing the binary exclusion graph of  $\phi_s$  over  $N_v^s$ . Excludes constraints that were not chosen to be represented as an Xor-group are added. When adding constraints, we control that the constraint is not already induced by the FM. At this end, it should be noted that the FM may still be an over approximation of  $\phi_s$ <sup>6</sup>. Using standard propositional logics techniques, we can calculate the complement between the current set of configurations represented by the FM and the expected set of configurations of the slice FM. The complement can be recovered, for instance, as a conjunction of propositional constraints.

#### 4.2.4 Properties and Comparison

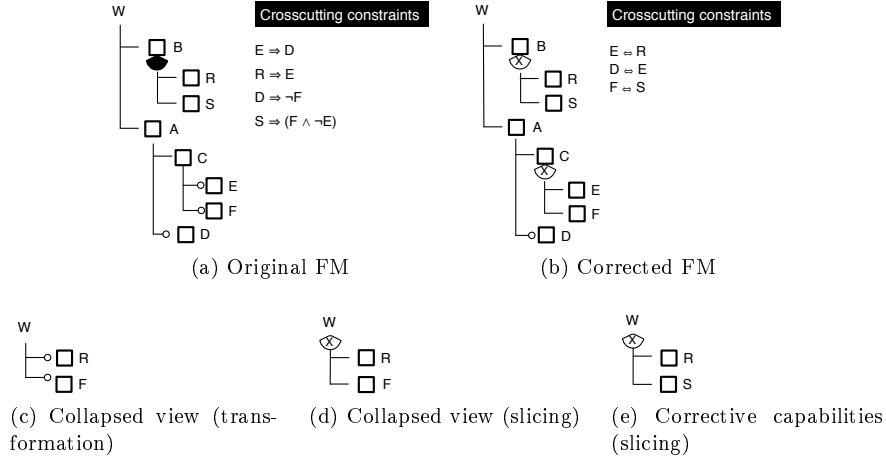
We now discuss properties of the slicing technique and the transformations described above regarding their ability to produce visualisations.

**Semantic preservation.** It is important to demonstrate that the visualisations preserve a form of semantic equivalence with the original FM. We define the semantic equivalence in terms of the set of configurations characterized by the original FM and the projected set of configuration characterized by the collapsed visualisation.

*Accuracy of visualisations.* As demonstrated in [36] for FMs without constraints, the greyed and pruned visualisations preserve the semantic equivalence. Using the syntactical transformations though, the semantic equivalence in the collapsed visualisation does not hold. Take the simple counter-example shown in Fig. 4a and the collapsed visualisation of view  $v$  depicted in Fig. 4c. A valid configuration of the collapsed visualisation would be  $\{W, R, S\}$ . However, that configuration is not valid in the FM since  $R$  and  $S$  must not appear together in a configuration. This shows that the transformation that produces the collapsed visualisation does not preserve the semantics of the FM: The collapsed visualisation is an under-constrained FM. This is, however, not a limitation in practice. When FBC is assisted by a solver, the solver preserves the global consistency of the FM. It thereby prevents possible errors induced by the under-constrained model presented in the collapsed visualisation. In the counter-example in Fig. 4c for instance, the selection of  $R$  in the view will automatically entail the deselection of  $S$ , even though the recomputed cardinality does not enforce that propagation.

<sup>6</sup> In [24], the authors characterized the limited expressiveness of FMs compared to propositional logic.

Using the slicing technique, the collapsed visualisation respects by construction the semantic equivalence. It exactly corresponds to Transformation 3 that specifies the relationship between the original FM and the slice FM. For example, the slicing is able to enforce that  $R$  and  $F$  features are mutually exclusive (see Fig. 4d).



**Fig. 4** Collapsed visualisations (transformation and slicing).

*Assumptions about input FMs.* In [36], the correctness of transformations for the three kinds of visualisations has been shown for FMs without cross-tree constraints. The reason is that arbitrary cross constraints can have an influence on the visualisations. By reasoning directly at the semantic level, the slicing technique is applicable to any kind of FMs, including arbitrary propositional constraints. However, the slicing technique does not support generic  $< i..j >$  cardinality.

*Corrective capabilities.* Due to the presence of constraints, an input FM may contain *anomalies*, for example, dead features, false optional features, or redundancies (see [12]). The slicing algorithm ensures, by construction, that there is no dead feature, correctly detect mandatory features and avoids redundancy of constraints. Therefore the slicing operator can be used as an *automated technique to correct* anomalies of FMs while preserving the original set of configurations and feature hierarchy. Two examples are given in Fig. 4b and Fig. 4e. Moreover the corrective modifications applied to the original FM can be detected and reported to SPL practitioner so that they can understand the anomalies.

*Impact on other kinds of visualisation.* Another application of the corrective property is that the slicing technique can be used to produce more accurate greyed and pruned visualisations (e.g. by correcting wrong cardinal-

ities). For the greyed visualisation, the slicing is first applied on the original FM, using the whole set of features as slicing criterion (see Fig. 4e for a corrected FM) while some features are then greyed out. For the pruned visualisation, the slicing is first applied on the original FM, using the set of features  $N_v^p$  of Definition 1 as slicing criterion. Features are then greyed out in line with the definition of pruned visualisation.

**Performance.** Synthesizing views at the semantic level, though more powerful, has a cost. Satisfiability techniques that reason over propositional formula are used and can be realized using either satisfiability (SAT) solvers or binary decision diagrams (BDDs) [24, 60].

*BDD-based implementation.* A BDD can be seen as a compact representation of a propositional formula. BDDs are known to efficiently compute the existential quantification of a propositional formula in at most polynomial time with respect to the sizes of the BDDs involved. Moreover, as shown in [24], BDDs can be used to synthesize an FM in polynomial time regarding the size of the BDD representing the input propositional formula. The primary limitation of a BDD-based implementation is related to the space complexity: (i) as shown in [21], computing the BDD of an FM containing more than 2000 features is intractable ; (ii) from our experiments, the synthesis of FMs has practical limits (up to  $800^7$  features) mainly due to the cost of computing Or-groups.

*SAT-based implementation.* BDDs do not scale for very large FMs (e.g. Linux FM that has more than 6000 features). In [60], She et al. proposed to rely on SAT solvers (rather than BDDs as in [24]) and reported that the use of SAT solvers is significantly more scalable. As SAT solvers require the formula to be in conjunctive normal form (CNF). To avoid the exponential explosion of disjunctive clauses, we developed specific techniques and some heuristics to determine the order in which existential quantification should be applied [4]. Using the slicing technique on generated and real-world FMs, we found that: (i) computing the propositional formula is almost instantaneous for all FMs of SPLOT (less than one second, whatever the size of the slicing criterion is); (ii) the SAT-based implementation scales for a number of features ( $\#features$ ) lesser than 10000 whatever the size of the slicing criterion is, but not for the Linux FM; (iii) the order in which the features are existentially quantified is of prior importance: We observe scalability issues when quantifying first the features that are at the top of the feature hierarchy for  $\#features \geq 2000$  ; (iv) for very large FMs ( $\#features \geq 5000$ ), the computation time is inadequate for an interactive use of the slice operator (up to 20 minutes).

**Summary and comparison.** On the one hand, the slicing technique is more general (i.e., applicable to any kind of FMs and propositional constraints) and accurate than the syntactical transformations. In particular the collapsed visualisation is no longer an under-approximation of the projected

---

<sup>7</sup> Janota et al. reported that the BDD-based algorithm proposed in [24] scales up only for FMs with 300/400 features [39], but did not use the heuristics proposed in [21] that reduce the size of BDDs.

set of configurations (see example in Fig. 4). On the other hand, some limitations remain: Lack of support for generalized cardinality, and performance issues. As a result, a *tradeoff* should be found when producing collapsed visualisations. In this case, the slicing or the syntactical transformations can be chosen on demand, i.e., regarding the kind of visualisations, the number of features, the presence of cross-tree constraints, etc. For other kinds of visualisations (greyed, pruned), anomalies can be first corrected, the syntactical transformations being applied afterwards. The consistency of the FM for under-constrained collapsed views can be maintained by reasoning about the complete FM in the back-end.

## 5 Tool support

Armed with these definitions, we now present two complementary tools that support view management. The first tool has been developed in the context of FBC while the second tool targets the large scale management of FMs through a dedicated language. They both support view *specification* with two similar solutions (i.e., XPath and a specific textual notation) and can be connected together.

### 5.1 View Creation and Visualisation in SPLOT

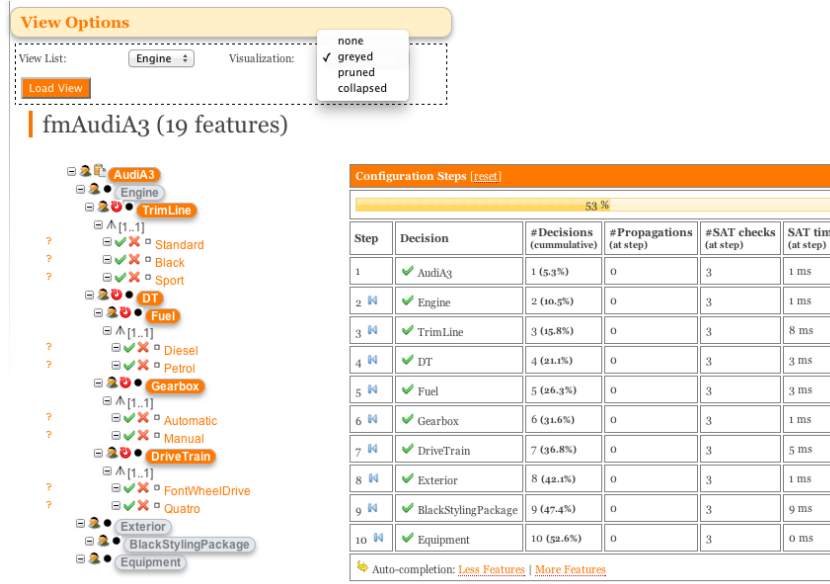
The tool support developed for multiview FBC builds upon SPLOT [49]. To provide efficient interactive configuration, SPLOT relies on a SAT solver (SAT4J) and a BDD solver (JavaBDD). Their reasoning abilities enable error detection and decision propagation. SPLOT was chosen because it offers robust support for FBC, it is easy to extend, and the existing repository of FMs is an excellent testbed for multiview FMs. All our extensions to SPLOT are available online.<sup>8</sup> The three extensions supporting multiview FBC are briefly introduced below.

The first extension enables view creation with XPath expressions. An online evaluator checks that the XPath expression is correct and shows the results of its evaluation. Moreover, the completeness of the views can be checked interactively and the features that are not covered, if any, are returned.

The actual configuration of a view is provided by the second extension. The extension allows stakeholders to select (1) the view to configure and (2) the visualisation. In Fig. 5, the view of the *Engine* is selected and the pruned visualisation is activated. Note the greyed *Exterior*, *Equipment* and *BlackStylingPackage* features that can neither be selected nor deselected.

<sup>8</sup> <http://www.splot-research.org/extensions/fundp/fundp.html>

The stakeholder can switch freely from one visualisation to another as she configures her view without loosing the decisions that were already made. This way, we *dynamically combine* the advantages of the three visualisations and leave complete freedom to the stakeholder to choose the one(s) that best fit(s) her preferences. The table on the right monitors the status of the current configuration. Basically, it tells what features have been selected or deselected, and which decisions were propagated. As explained in Section 4.2, the solver reasons about the full FM and not only about an individual view. Thereby, the decision to select or deselect a feature in the view is propagated in the complete model—keeping the global configuration consistent.

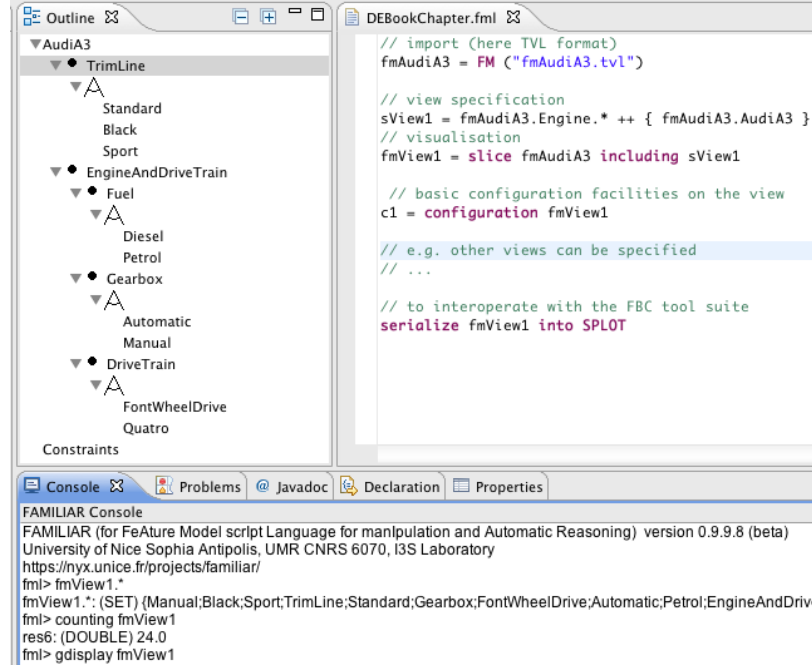


**Fig. 5** Configuration view of the *Engine* with the greyed visualisation in SPLOT.

The third extension provides basic support for multi-user concurrent configuration. At the time being, it only enables synchronous configuration. To prevent conflicting decisions, a configuration session manager is used. Its role is (1) to maintain a mutual exclusion on the configuration engine so that only one user can commit a decision at a time, and (2) to notify all users about a decision and about the results of the propagation.

## 5.2 Slicing and FAMILIAR

As seen in the previous sections, the slicing operator can be used to produce collapsed visualisations. The operator is part of FAMILIAR (for *FeAture Model sCriPt Language for manIpulation and Automatic Reasoning*) a domain-specific language for large scale management of FMs [3].



**Fig. 6** Slicing example and interoperability in the FAMILIAR environment.

Examples of the syntax of the slicing operator are given in Fig. 6. The set of features that constitutes the slicing criterion can be specified either by inclusion (keyword: `including`) or exclusion (keyword: `excluding`). Intentional definitions of views can be specified in the style of XPath. Off-the-shelf SAT solvers (i.e., SAT4J) and BDD library (i.e., JavaBDD) are internally used. The slicing operator produces a new FM that can be manipulated using variables. FAMILIAR also includes functions for composing FMs, editing FMs (e.g., renaming and removal of features), reasoning about FMs (e.g., validity, comparison of FMs) and their configurations (e.g., counting or enumerating the configurations in an FM). FAMILIAR comes with an Eclipse-based environment that is composed of textual editors, an interpreter that executes FAMILIAR scripts, and an interactive front-end. The FMs can be serialized in different formats (SPLOT, FeatureIDE, a subset of TVL, etc.).

Thanks to the integration with SPLOT, we can realize scenarios in which an FM is first corrected using the slicing operator of FAMILIAR and then used in the FBC web environment.

## 6 Conclusion

Feature models (FMs) are widely used to represent the valid combination of features supported by a family of systems in a given domain. In real variability-intensive systems, many *concerns* have to be considered. These concerns are related in a variety of ways and there can be thousands of features whose legal combinations are governed by many and often complex rules. It has been observed that configuring or maintaining a single large FM may not be feasible [54, 26]. Views (as a simplified representation of an FM tailored for a specific stakeholder, role or task) have been repeatedly identified as a possible solution to the scalability and configuration issues of FMs.

In this chapter, we reviewed concerns and their separation in FMs, revisiting the concept of view, and discussing the major results in the literature. Then, we delved into the three specific problems of multi-view FMs: The *specification* of a view, the *coverage* of a set of views, and the *visualisation* of a view. Finally, we presented two tools that provide support for these three problems.

Several avenues for future work can be envisaged. The three alternative visualisations were developed to provide more flexibility to the configuration environment and more precise contextual information to the user. That improvement is, however, limited to tree-like representations of FMs. Recent advances deviate from the traditional explorer-like representations [15, 17], while others recommend dedicated configuration interfaces [52, 16]. Understanding the most suitable interfaces for multi-view will require qualitative user studies. More generally, we plan to study further the practical usage and applicability of the proposed techniques in various domains (e.g., operating systems [13, 66] and video surveillance [6]).

## References

1. Mathieu Acher. *Managing Multiple Feature Models: Foundations, Language and Applications*. PhD thesis, University of Nice Sophia Antipolis, Nice, France, 2011.
2. Mathieu Acher, Philippe Collet, Philippe Lahire, and Robert France. Comparing approaches to implement feature model composition. In *Proceedings of the 6th European Conference on Modelling Foundations and Applications (ECMFA'10)*, volume 6138 of *LNCS*, pages 3–19, 2010.
3. Mathieu Acher, Philippe Collet, Philippe Lahire, and Robert France. A domain-specific language for managing feature models. In *Proceedings of the Symposium on Applied Computing (SAC'11)*, pages 1333–1340. ACM, 2011.

4. Mathieu Acher, Philippe Collet, Philippe Lahire, and Robert France. Separation of Concerns in Feature Modeling: Support and Applications. In *Proceedings of the 11th International Conference on Aspect-Oriented Software Development (AOSD'12)*, pages 1–12. ACM, 2012.
5. Mathieu Acher, Philippe Collet, Philippe Lahire, Alban Gaignard, Robert France, and Johan Montagnat. Composing multiple variability artifacts to assemble coherent workflows. *Software Quality Journal (Special issue on Quality for SPLs)*, 2011. <http://dx.doi.org/10.1007/s11219-011-9170-7>.
6. Mathieu Acher, Philippe Collet, Philippe Lahire, Sabine Moisan, and Jean-Paul Rigault. Modeling variability from requirements to runtime. In *ICECCS'11*, pages 77–86. IEEE, 2011.
7. Mathieu Acher, Patrick Heymans, Philippe Collet, Clément Quinton, Philippe Lahire, and Philippe Merle. Feature Model Differences. In *Proceedings of the 24th International Conference on Advanced Information Systems Engineering (CAiSE'12)*, LNCS. Springer, 2012.
8. M. Antkiewicz and K. Czarnecki. FeaturePlugin: feature modeling plug-in for Eclipse. In *Proceedings of the 2004 OOPSLA workshop on eclipse technology eXchange*, pages 67–72, Vancouver, BC, Canada, 2004. ACM.
9. Sven Apel and Christian Kästner. An overview of feature-oriented software development. *Journal of Object Technology (JOT)*, 8(5):49–84, July/August 2009.
10. D. Batory, J Liu, and J. N. Sarvela. Refinements and multi-dimensional separation of concerns. In *Proceedings of the 9th European Software Engineering Conference (ESEC'03) held jointly with FSE'03*, pages 48–57. ACM, 2003.
11. Don Batory, Jia Liu, and Jacob Neal Sarvela. Refinements and multi-dimensional separation of concerns. *SIGSOFT Softw. Eng. Notes*, 28:48–57, 2003.
12. D. Benavides, S. Segura, and A. Ruiz-Cortes. Automated analysis of feature models 20 years later: A literature reviews. *Information Systems*, 35(6), 2010.
13. T. Berger, S. She, R. Lotufo, A. Wasowski, and K. Czarnecki. Variability modeling in the real: a perspective from the operating systems domain. In *Proceedings of the 25th International Conference on Automated Software Engineering (ASE'10)*, pages 73–82. ACM, 2010.
14. C. Bidian. From stakeholder goals to product features: towards a role-based variability framework with decision boundary. In *Proceedings of the 4th International Conference on Privacy, Security and Trust (PST '06)*, pages 1–5. ACM, 2006.
15. G. Botterweck, S. Thiel, D. Nestor, S. bin Abid, and C. Cawley. Visual tool support for configuring and understanding software product lines. In *Proceedings of the 12th International Software Product Line Conference (SPLC '08)*, pages 77–86. IEEE, 2008.
16. Quentin Boucher, Gilles Perrouin, and Patrick Heymans. Deriving configuration interfaces from feature models: a vision paper. In *Sixth International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS'12)*, pages 37–44, 2012.
17. C. Cawley, P. Healy, G. Botterweck, and S. Thiel. Research tool to support feature configuration in software product lines. In *Proceedings of the 4th International Workshop on Variability Modelling of Software-intensive Systems (VaMoS'10)*, pages 179–182. University of Duisburg-Essen, January 2010.
18. D. Clarke and J. Proenca. Towards a theory of views for feature models. In *Proceedings of the 1st International Workshop on Formal Methods in Software Product Line Engineering (FMSPLE'10)*, 2010.
19. A. Classen, P. Heymans, and P.-Y. Schobbens. What's in a Feature: A Requirements Engineering Perspective. In *FASE'08, held as Part of ETAPS'08*, pages 16–30, 2008.
20. A. Classen, A. Hubaux, and P. Heymans. A formal semantics for multi-level staged configuration. In *Proceedings of the 3rd International Workshop on Variability Modelling of Software-intensive Systems (VaMoS'09)*, pages 51–60, 2009.



21. K. Czarnecki, S. She, and A. Wasowski. Sample spaces and feature models: There and back again. In *Proceedings of the 12th International Software Product Line Conference (SPLC'08)*, pages 22–31. IEEE, 2008.
22. Krzysztof Czarnecki, Simon Helsen, and Ulrich W. Eisenecker. Formalizing cardinality-based feature models and their specialization. *Software Process: Improvement and Practice*, 10(1):7–29, 2005.
23. Krzysztof Czarnecki, Simon Helsen, and Ulrich W. Eisenecker. Staged configuration through specialization and multi-level configuration of feature models. *Software Process: Improvement and Practice*, 10(2):143–169, 2005.
24. Krzysztof Czarnecki and Andrzej Wasowski. Feature diagrams and logics: There and back again. In *Proceedings of the 13th International Software Product Lines Conference (SPLC'07)*, pages 23–34. IEEE, 2007.
25. Sybren Deelstra, Marco Sinnema, and Jan Bosch. Product derivation in software product families: a case study. *J. Syst. Softw.*, 74(2):173–194, 2005.
26. Deepak Dhungana, Paul Grünbacher, Rick Rabiser, and Thomas Neumayer. Structuring the modeling space and supporting evolution in software product line engineering. *Journal of Systems and Software*, 83(7):1108–1122, 2010.
27. eCos. User Guide. <http://ecos.sourceware.org/docs-latest/user-guide/ecos-user-guide.html>, March 2011.
28. A. Finkelstein, J. Kramer, B. Nuseibeh, L. Finkelstein, and M. Goedicke. Viewpoints: A framework for integrating multiple perspectives in system development. *International Journal on Software Engineering and Knowledge Engineering*, 2:31–58, 1992.
29. M. Glinz, , and R. J. Wieringa. Guest editors' introduction: Stakeholders in requirements engineering. *IEEE Software*, 24:18–20, 2007.
30. O. Gotel and A. Finkelstein. Contribution structures. In *Proceedings of the 2nd International Conference on Requirements Engineering (RE'95)*, pages 100–107. IEEE, 1995.
31. P. Grünbacher, R. Rabiser, D. Dhungana, and M. Lehofer. Structuring the product line modeling space: Strategies and examples. In *Proceedings of the 3rd International Workshop on Variability Modelling of Software-intensive Systems (VaMoS'09)*, pages 77–82, 2009.
32. Herman Hartmann and Tim Trew. Using feature diagrams with context variability to model multiple product lines for software supply chains. In *Proceedings of the 12th International Software Product Lines Conference (SPLC'08)*, pages 12–21. IEEE, 2008.
33. Herman Hartmann, Tim Trew, and Aart Matsinger. Supplier independent feature modelling. In *Proceedings of the 13th International Software Product Lines Conference (SPLC'09)*, pages 191–200. IEEE, 2009.
34. Hubaux. *Feature-based Configuration: Collaborative, Dependable, and Controlled*. PhD thesis, University of Namur, Belgium, 2012.
35. A. Hubaux, A. Classen, and P. Heymans. Formal modelling of feature configuration workflow. In *Proceedings of the 13th International Software Product Lines Conference (SPLC'09)*, pages 221–230, San Francisco, CA, USA, 2009. Carnegie Mellon University.
36. A. Hubaux, P. Heymans, P.-Y. Schobbens, D. Deridder, and E. Abbasi. Supporting multiple perspectives in feature-based configuration. *Software and Systems Modeling (SoSyM)*, pages 1–23, 2011.
37. Arnaud Hubaux, Patrick Heymans, Pierre-Yves Schobbens, and Dirk Deridder. Towards multi-view feature-based configuration. In *16th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ'10)*, volume 6182 of *LNCS*, pages 106–112. 2010.
38. M. Janota. *SAT Solving in Interactive Configuration*. PhD thesis, University College Dublin, 2010.
39. Mikolás Janota, Victoria Kuzina, and Andrzej Wasowski. Model construction with external constraints: An interactive journey from semantics to syntax. In *11th International Conference on Model Driven Engineering Languages and Systems (MODELS'08)*, volume 5301 of *LNCS*, pages 431–445, 2008.

40. K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, 1990.
41. K. C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh. Form: A feature-oriented reuse method with domain-specific reference architectures. *Annals of Software Engineering*, 5:143–168, 1998.
42. C. Krueger. BigLever Software, Inc. <http://www.biglever.com/index.html>, May 2010.
43. Jérôme Lang and Pierre Marquis. On propositional definability. *Artificial Intelligence*, 172(8-9):991–1017, 2008.
44. M. Mannion, J. Savolainen, and T. Asikainen. Viewpoint-oriented variability modeling. In *Proceedings of the 33rd International Computer Software and Applications Conference (COMPSAC'09)*, pages 67–72. IEEE, 2009.
45. M. Mendonça. SPLOT. <http://www.splot-research.org/>, May 2010.
46. M. Mendonça, T. Tonelli Bartolomei, and D. Cowan. Decision-making coordination in collaborative product configuration. In *Proceedings of the 23rd Symposium on Applied computing (SAC'08)*, pages 108–113, Fortaleza, Ceara, Brazil, 2008. ACM.
47. Marcílio Mendonça. *Efficient Reasoning Techniques for Large Scale Feature Models*. PhD thesis, University of Waterloo, 2009.
48. Marcílio Mendonça, Donald D. Cowan, William Malyk, and Toacy Cavalcante de Oliveira. Collaborative product configuration: Formalization and efficient algorithms for dependency analysis. *Journal of Software*, 3(2):69–82, 2008.
49. M. Mendonça, M. Branco, and D. Cowan. S.P.L.O.T.: software product lines online tools. In *Proceeding of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications (OOPSLA'09)*, pages 761–762. ACM, 2009.
50. A. Metzger, P. Heymans, K. Pohl, P.-Y. Schobbens, and G. Saval. Disambiguating the documentation of variability in software product lines: A separation of concerns, formalization and automated analysis. In *Proceedings of 15th International Conference on Requirements Engineering (RE'07)*, pages 243–253. IEEE, 2007.
51. B. Nuseibeh, J. Kramer, and A. Finkelstein. Viewpoints: meaningful relationships are difficult! In *Proceedings of the 25th International Conference on Software Engineering (ICSE'03)*, pages 676–681. IEEE, 2003.
52. A. Pleuss, G. Botterweck, and D. Dhungana. Integrating Automated Product Derivation and Individual User Interface Design. In *Proceedings of the 4th International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS'10)*, pages 69–76, 2010.
53. Klaus Pohl, Gunter Bockle, and Frank van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer, July 2005.
54. Klaus Pohl, Günter Böckle, and Frank J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag, 2005.
55. pure-systems GmbH. Variant management with pure::variants. <http://www.pure-systems.com/fileadmin/downloads/pv-whitepaper-en-04.pdf>, 2006.
56. M.-O. Reiser and M. Weber. Managing highly complex product families with multi-level feature trees. In *Proceedings of the 14th International Conference on Requirements Engineering (RE'06)*, pages 146–155. IEEE, 2006.
57. Mark-Oliver Reiser and Matthias Weber. Multi-level feature trees: A pragmatic approach to managing highly complex product families. *Requir. Eng.*, 12(2):57–75, 2007.
58. P.-Y. Schobbens, P. Heymans, J.-C. Trigaux, and Y. Bontemps. Generic semantics of feature diagrams. *Computer Networks*, 51(2):456–479, 2007.
59. Pierre-Yves Schobbens, Patrick Heymans, Jean-Christophe Trigaux, and Yves Bontemps. Feature Diagrams: A Survey and A Formal Semantics. In *Proceedings of the 14th International Requirements Engineering Conference (RE'06)*, pages 139–148. IEEE, 2006.
60. S. She, R. Lotufo, T. Berger, A. Wasowski, and K. Czarnecki. Reverse engineering feature models. In *Proceedings of the 33th International Conference on Software Engineering (ICSE'11)*, pages 461–470. ACM, 2011.

61. P. Tarr, H. Ossher, W. Harrison, and S. M. Jr. Sutton. N degrees of separation: multi-dimensional separation of concerns. In *Proceedings of the 21st International Conference on Software Engineering (ICSE'99)*, pages 107–119. IEEE, 1999.
62. J. M. Thompson and M. P.E. Heimdahl. Structuring product family requirements for n-dimensional and hierarchical product lines. *Requirements Engineering Journal*, 8(1):42–54, 2003.
63. T. T. Tun, Q. Boucher, A. Classen, A. Hubaux, and P. Heymans. Relating requirements and feature configurations: A systematic approach. In *Proceedings of the 13th International Software Product Lines Conference (SPLC'09)*, pages 201–210. IEEE, 2009.
64. Thein Than Tun and Patrick Heymans. Concerns and their separation in feature diagram languages - an informal survey. In *Workshop on Scalable Modelling Techniques for Software Product Lines (SCALE@SPLC'09)*, pages 107–110, 2009.
65. Mark Weiser. Program slicing. In *Proceedings of the International Conference on Software Engineering (ICSE '81)*, pages 439–449. IEEE, 1981.
66. Y. Xiong, A. Hubaux, S. She, and K. Czarnecki. Generating range fixes for software configuration. In *Proceedings of the 34th International Conference on Software Engineering (ICSE'12)*. IEEE, 2012. 58-68.
67. H. Zhao, W. Zhang, and H. Mei. Multi-view based customization of feature models. *Journal of Frontiers of Computer Science and Technology*, 2(3):260–273, 2008.