A Visual Support for Decomposing Complex Feature Models

Simon Urli*, Alexandre Bergel†, Mireille Blay-Fornarino*, Philippe Collet*, and Sébastien Mosser*
*Université Nice – Sophia Antipolis , CNRS - I3S - UMR 7271, 06900 Sophia Antipolis, France
† PLEIAD Lab, Department of Computer Science (DCC), University of Chile, Chile

***Abstract*—In Software Product Line (SPL) engineering, Feature Models (FMs) are widely used to capture and manage variability in a sound and organized fashion. Though semantics, notations and reasoning support are well established, maintaining large FMs is still an open problem. As large FMs naturally contain different concerns, some related to domains, others being inherently cross-cutting ones, it is challenging to find a decomposition that will tame this complexity and ease maintenance.**

This paper presents a visual representation of dependent FMs useful in decomposing a large FM while quantitatively visualizing constraints between and inside them. This *Variability Blueprint* is intuitive enough to enable the SPL maintainer to confine dependencies between FMs in a small set of identified features inside each decomposed FM. We describe our blueprint and report on its application on two case studies.

***Index Terms*—Feature Modeling, Visualization, Composition, Software Product Lines.**

I. Introduction

Reusing artifacts is central to the software engineering discipline. The Software Product Lines (SPL) approach advocates the development of software system families rather than individual software products, aiming at a paradigm shift to support mass customization through systematized reuse [1]. SPL engineering is about building and maintaining similar software products within an application domain, exploiting common parts and managing variable ones among products. In this context the modeling and management of variability is a central activity. *Feature models* (FMs) are then widely used to model the variability of a domain or system, in terms of mandatory, optional and exclusive features organized in a hierarchy, as well as propositional *cross-tree* constraints over these features [2], [3]. Developments around formal semantics, reasoning techniques and tool support [3], [4], [5] make FM a *de facto* standard for managing variability, largely accepted in industry. Some practical issues have been identified in a recent study [6]: very large FMs are observed (10,000 features), several FMs are manipulated in the same project, cross-tree constraints are frequent and make FMs complex to understand and maintain. This study also shows that practitioners identify both visualization and evolution of a model as the two most important challenges.

In order to ease comprehension and maintenance of large and complex FMs, previous work has proposed composition operators (aggregation, merge) for FMs [7], as well as a decomposition operator to *slice* a FM according to a given set of features [8]. These operators are semantically well-founded and efficiently implemented by solving techniques so to handle large FMs. Decomposing FMs is thus possible, and especially desirable as maintaining a single large FM for an entire system may not be feasible [9], [10], [11], [12].

Our experience in the development of several real-world SPLs in different domains [7], [13] and similar organization of fragmented SPLs [10] is evidence that composing and decomposing FMs helps in managing large and complex SPLs. Even with decomposed smaller FMs, visualization issues are, however, still present, as remaining FMs are of a large size, and relations between features inside a FM and between related FMs are hard to grasp. Several research results [14], [15], [16], [17], [18] have produced variability visualizations that were more expressive than the traditional FODA feature diagram [2]. However, they only focus on interactive configuration processes and cannot be applied to large inter-related FMs during maintenance phases.

This paper explores the use of visualization to cope with the inherent complexity of large and related feature models. We present *Variability Blueprint*, a polymetric-based visualization [19] that contrasts complexity metrics against both the hierarchical structure of a feature model and its internal and external constraints when it is related to other feature models. We evaluate our blueprint on FraSCAti, an open-source component assembly middleware platform that is represented by a complex medium-size feature model. Our case study indicates that Variability Blueprint is a reliable support to monitor large feature model decompositions and reengineering.

We employ polymetric view as the foundation of our approach. The reason for this choice stems from empirical evidence that shows this technique is an effective option for software visualizations [20], [21], [22].

Our visualization has been implemented in the Moose software analysis platform[1] and our visualization is available under the MIT license[2].

This paper is structured as follows. Section II encompasses themes of visualization and complexity in feature modeling that served as motivation for our research. Section III describes Variability Blueprint, a visual representation of large feature models inter-related by constraints. Section IV reports on the application of our representation on two different case studies that are based on real SPLs. Section V discusses related work

[1] http://moosetechnology.org
[2] http://smalltalkhub.com/#!/~abergel/Familiar

while Section VI concludes this paper and describes future work.

II. Motivation

A. Visualizing Feature Models

A feature model (FM) [2] defines a *hierarchy*, structuring features into levels of increasing detail, as well as the variability itself, which is expressed on features and through cross-tree constraints. In the hierarchy, when a feature is composed of subfeatures, these subfeatures may be *optional*, *mandatory* or define a group (XOR or OR). In addition, any *propositional* constraints (e.g., implies or excludes) can be specified to express complex dependencies between features. A FM defines a set of valid feature configurations. They are obtained by selecting features in the FM, so that *i)* if a feature is selected, its parent is also selected; *ii)* if a parent is selected, all the mandatory subfeatures, exactly one subfeature in each of its XOR-groups, and at least one of its OR groups are selected; *iii)* propositional constraints hold.

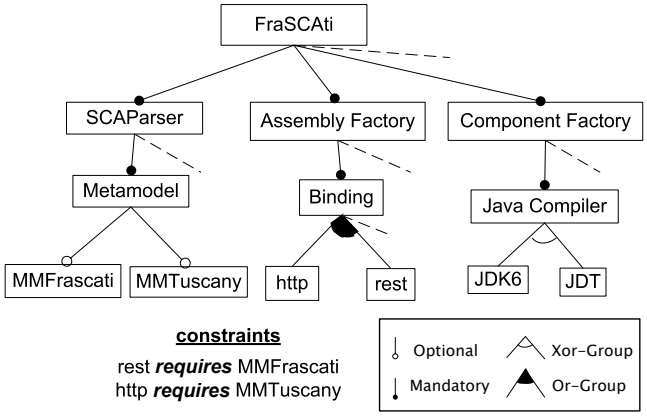

Fig. 1: A Feature Diagram

FMs were introduced in the FODA method [2], which also provided a graphical representation through feature diagrams. An example extracted from our FraSCAti platform case study is depicted in Figure 1. While textual representations for editing and manipulating FMs have also been proposed, some recent controlled experiments [23] show the advantages of graphical representation to increase cognitive efficiency and effectiveness of novice practitioners, as well as quality of produced FMs.

Much effort has been made on variability visualization with a focus on visualizing a FM during its interactive configuration [14], [15], [16], [18] (*i.e.,* selecting a feature and showing impacts with deselected features). Specific visualizations have been proposed, combining techniques such as color-coding (*i.e.,* green for selected features, red for deselected ones) [14], details on demand and focus with zoom on a chosen part of the FM hierarchy [15], [16]. Different kinds of visualization techniques, ranging from cone trees to Venn diagrams, have been envisaged [17], but none that can be applied to large inter-related FMs during maintenance phases, as they do not scale well to large FMs and do not represent constraints in and between FMs at the same time.

B. Feature Model Complexity

The ISO 9126 model for software product quality describes maintainability as one of the 6 main characteristics of software product quality. It is characterized by a set of attributes that bear on the effort needed to make specified modifications, among others:

- analyzability is the capability of the conceptual model of a software system to be diagnosed for deficiency;
- changeability is the possibility and ease of change in a model when modifications are necessary;
- understandability is the prospect and likelihood of the software system model to be understood and comprehended by its users or other model designers.

Analyzability is possible on a single FM, as FMs have been semantically related to propositional logic [4]. Their configuration set can be described by a propositional formula, which allows for automating FM analysis [5] to detect defects and anomalies, *e.g.,* dead features, void feature models. Changeability is partially achieved by reasoning on feature edits [24] so to reason on the modifications (refactoring, specialization, generalization or arbitrary edit of the FM). Still, understanding a non-trivial FM may be challenging in presence of numerous and constrained features.

A number of dedicated metrics measure feature model complexity and relate such complexity to maintainability [25]:

- *Number of Features (NF)*: The total number of features in a FM.
- *Number of Leaf Features (NLeaf)*: The number of features with no children or further specializations (*i.e.,* the leaf of the FM tree).
- *Cyclomatic Complexity (CC)*: The number of distinct cycles that can be found in the FM. As a FM is a tree, cycles can only be caused by cross-tree constraints between features. It is simple to show that the number of distinct cycles and hence cyclomatic complexity of a FM is equivalent to the number of its constraints.
- *Cross-Tree Constraints (CTC)*: The ratio of the number of unique features involved in the FM cross-tree constraints over the number of features in the FM. This measure represents the degree of involvement of features in the definition of cross-tree constraints.
- *Flexibility of Configuration (FoC)*: The ratio of the number of optional features over the number of features in the FM. The rationale behind this metric is that the more optional features exist in a FM, the more choices are available for the designers to choose from while configuring the FM.
- *Number of Valid Configurations (NVC)*: The number of all possible and valid configurations that can be derived from the FM w.r.t. its tree structure and cross-tree constraints.

The set {NLeaf, CC, NVC, FoC} was identified to be the sufficient subset of the proposed metrics for evaluating maintainability of a single FM [25]. This study was however

conducted on small to medium-size FMs (with an average of 76 features per FM) while the current practice shows that cross-tree constraints are widely used in FMs [6] and that they strongly influence SAT-based analysis algorithms on large FMs [26]. In addition, with large FMs, its size in terms of total number of features (NF) is also likely to influence maintainability. We thus consider NF and CTC important metrics when dealing with the maintainability of large and complex FMs.

C. Decomposing Feature Models: How Visualization Can Help

As previously mentioned, both the decomposition and composition of FMs [8], [7] are efficient mechanisms to tame the complexity of large and complex FMs [10], [7], [13]. Decomposing FMs can have different purposes, as FMs may capture variabilities at different levels of abstraction [27], [9], from several concerns [28], and from external parties [29], [27].

Whatever the final purpose, a semantically correct decomposition of a FM into related FMs should keep the semantics of the cross-tree constraints from the original FM into the resulting FMs. Taking the example of a common *implies* constraint $A \implies B$ between features A and B, can lead to two cases:

- A and B are kept in the same FM when the original one is decomposed. We define this constraint as *internal*.
- A and B are put into two separated FMs during decomposition, so their constraint is kept as an *external*, or *inter-FM*, constraint.

Depending on how the decomposition is carried out (which features are put in which FMs), the resulting FMs may be related to many inter-FMs constraints. As each decomposed FM is aimed at being as comprehensible as possible on its own, these inter-FMs constraints are creating some useless complexity. In several large and complex SPLs [7], [13], we observed that this inter-FMs constraints decreases maintainability. In this case, visualization can help by providing a crisp representation of internal and external constraints. Currently the common FM visualizations [16], [18] are not well-suited, as they mainly use curved lines to link the two constrained features, in our case leading to intermingled lines.

In addition, even with decomposed FMs, they are still large, so that visualizing the whole FM does not require the level of detail of a feature diagram. For example, feature names are less important, whereas knowing which features are the most involved in inter or intra constraints is extremely relevant to our maintainability objective. It is also important to keep some strong semantic information on the FM itself; its hierarchy and the type of nodes (*e.g.,* optional, mandatory) are salient information to understand how variability is organized within a FM.

Our aim is not to propose a method to determine whether a decomposition is relevant for a specific domain or stakeholder, or whether the *concerns* are well decomposed (if this is ever possible). Instead, our objective is to provide an appropriate visualization when a SPL maintainer has to determine what complexity is produced by applying a possible decomposition.

III. Variability Blueprint

Our first step toward decomposing complex feature models is to provide a meta-model that is rich enough to express quality attributes presented earlier. As the decomposition of FMs is following the general principle of separation of concerns [7], we use these concepts to organize our meta-model around two models. As depicted in Figure 2, the SPL part defines a *domain model* with a set of *Concern*s. Each concern abstracts a configuration space (*i.e.,* a Feature Model capturing some variability). Constraints between concerns (*External Constraint*s) involve features from different concerns (*i.e.,* decomposed FMs).

We specialize some of the FM metrics to quantify dependencies between FMs:

- *external Cyclomatic Complexity (exCC)* is the number of distinct cycles that can be found between two feature models (due to inter-FMs constraints). This is the external counterpart of the CC metric.
- *external Cross-Tree Constraints (exCTC)* is the ratio of the number of unique features involved in the external constraints over the number of features in the FM. While the CTC measures an intra-constraints ratio, exCTC represents the degree of involvement of features in the definition of the inter-FMs constraints. We expect it to be low if these constraints are gathered on few features.

Blueprint description. *Variability blueprint* is a visual representation of a FM. Our blueprint visually represents feature hierarchies while emphasizing the constraints, being internal or external, associated to each feature. An example of the blueprint is given in Figure 3.

Each box represents a feature. As with traditional feature visualizations, an edge describes the parent relationship between two features – a child feature being located below its parent feature. However, in order to make large representation more readable, the kind of a sub-feature (*e.g.,* optional, mandatory) is expressed by a color-coding on the edge, *e.g.,* blue for optional, green for mutual exclusion. The shape of a feature is indicated by two metrics:

- *width* indicates the number of external constraints in which this feature is involved.
- *height* indicates the amount of constraints internal to the concern/decomposed FM in which the feature is involved.

Features may also visually express some patterns. In Figure 3 several sub-features of feature `A` have both numerous internal and external constraints. These features therefore constrain other features both externally and internally. With such a pattern of internal and external constraints distributed among numerous features, it is difficult to understand this part (or concern) of the FM semantics, as impacts on the outside are not directly exposed. On the contrary, all sub-features of `B` are tall and thin as these features have numerous internal constraints and no external constraints. This shows that `B` can be configured internally. Finally looking at `A` and `B` at the same time, we can say that `A` might act as a kind of proxy from the whole FM

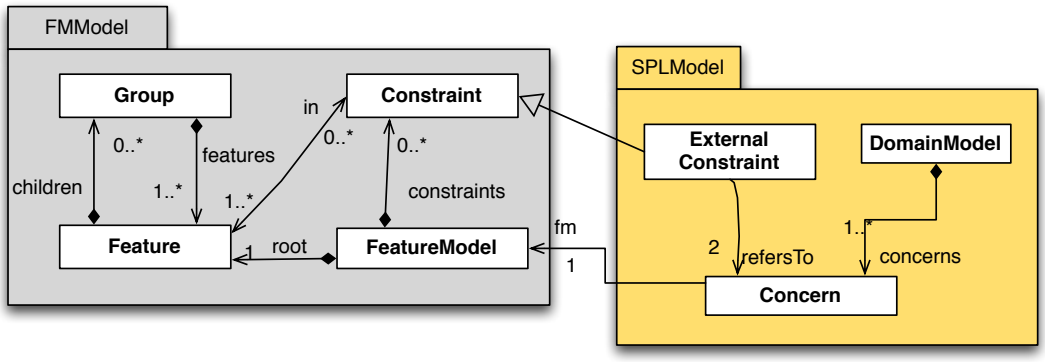

Fig. 2: Metamodel used to represent feature models broken down in multiple concerns

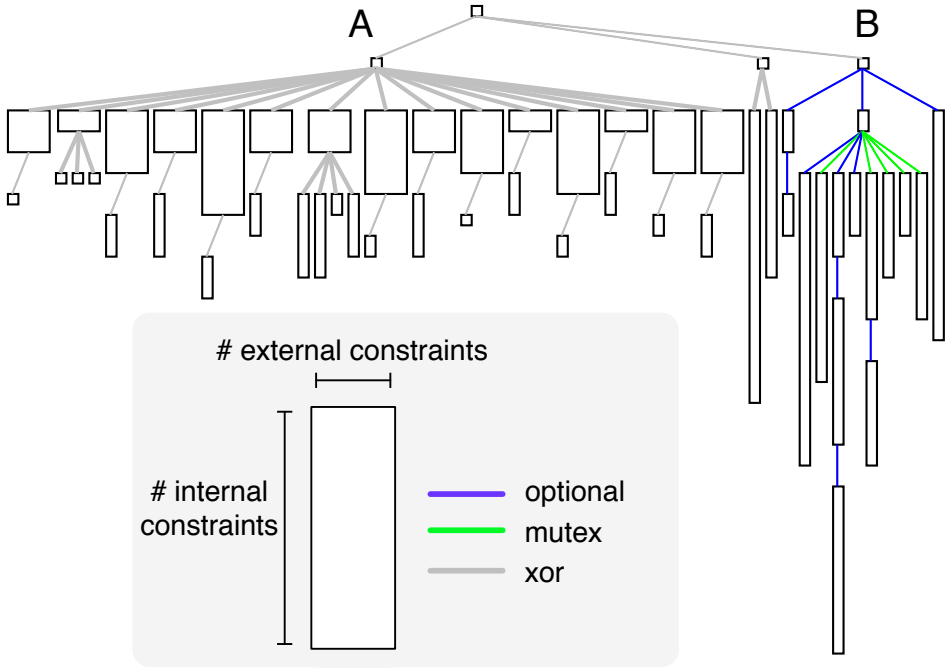

Fig. 3: Variability blueprint example

of the figure to the outside as it has all the features that have external constraints. At this point, it is however not possible to know whether the external constraints related to A:

- may transitively trigger constraints on features of B,
- may be restricted to other features of A,
- or a mix of both.

A SPL maintainer will have to decompose A according to some rules or identified concerns of the domain to visualize the results in a new blueprint showing the decomposed FMs from A and B. This case illustrates the kind of scenarios that we have experimented on industrial SPLs and are reported in the next section.

Interaction. Variability Blueprint has been designed to efficiently indicate the structure of FMs. The presence of constraints shape each feature, however dependencies between features are not indicated on the blueprint. The rationale for this is to not overload the blueprint while still relying on its structure. To be effective, the visualization has to provide ways to dig into features, which is supported by some interactions. For example, moving the mouse cursor over a feature highlights some related features (cf. Figure 4). Clicking on a feature opens an inspector detailing it, notably with its name, the number of internal constraints and the number of features that may affect it (*i.e.,* feature that can select or deselect, by means of a constraint or an XOR-group)[3]. Features may be drag-and-dropped and searched for using regular expression. The whole

[3]Weighted Internal Constraints and Launch are values that are related to the configuration process and are out of the scope of this paper.

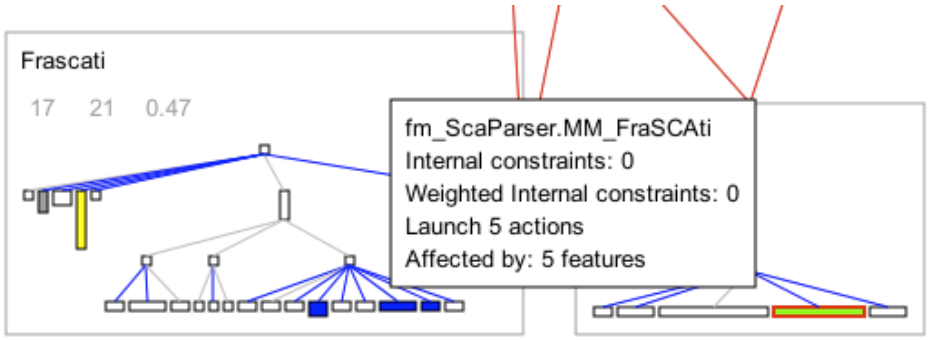

Fig. 4: Available interactions on a feature

visualization may be zoom-in and out are is exportable in numerous file formats.

Implementation. Our visualization support is available under the MIT license[4]. It has been implemented with the Moose software analysis platform (http://moosetechnology.org), while the decomposition of feature models and the maintenance of their dependencies rely on both the FAMILIAR domain-specific language [7][5] and the SpineFM framework [30][6].

IV. Case Studies

To illustrate and show the applicability of Variability Blueprint, we report its application on two different industrial case studies. The first one aims at finding a decomposition of a large FM with less complexity. The second one aims at visualizing and understanding a complex SPL in which concerns have been decomposed from the start.

A. Decomposing a Large FM

Our first case study concerns FraSCAti [31], an open-source component assembly middleware platform. A feature model representing all available FraSCAti configurations has been created using reverse engineering techniques on its architecture and plugin dependencies [32]. This operation was conducted through 6 different versions, noted V0 until V5, corresponding to different decompositions of the original FM. Different versions have been manually identified by a domain expert.

This section summarizes the different versions, their motivations, and the impact on Variability Blueprint. Figure 5 provides a visual evolution of these versions (V2 and V3 are not represented, to save space).

V0 → V1. The reverse engineered information from the FraSCAti architecture is initially gathered in a monolithic FM (V0): visually, this is represented with V0 in Figure 5 being a unique large feature model.

Different cohesive parts of the original feature hierarchy have been identified and moved into dedicated FMs with the help of the main architect of FraSCAti. This first decomposition attempts to distinguish high-level functionalities of the configurable middleware platform:

- The `ScaParser` part loads software components following the Service Component Architecture (SCA) standard [33]. As the standard is extensible and supports different implementations, the related variability is supposed to be captured in this part.
- `AssemblyFactoryImplementation` checks SCA components and drive their instantiation. Numerous plugins are combined in this factory according to component implementations, interface definition languages and service bindings (*e.g.,* http, REST).
- `ComponentFactory` actually instantiates the SCA components. Variability related to different Java compilers and containers is supposed to be captured in this part.
- The `Frascati` part presents a high-level view of the middleware, shielding the non-specialist user from the implementation details contained in the three other FMs.

The first and second lines of Table I give some metrics for the first two considered versions[7]. The number of features and constraints overall are first given. Then the details on all FMs of the four metrics we consider (CC and CTC for internal constraints, exCC and exCTC for external inter-FMs constraints) are shown. We observe that while smaller FMs have been obviously created by the decomposition, an important complexity is present in the dependencies between these FMs, with exCC being 17, 21 and 16, and exCTC being 47%, 54% and 71% for three of the FMs in V1.

V1 → V4. We study here several steps with the same objective. On some versions depicted in Figure 5, we have pinpointed several features by lower-case letters (**a** to **g**). In V1, nodes **b** and **c** depend on **a** with the two relations $\mathbf{b} \implies \mathbf{a}$ and $\mathbf{c} \implies \mathbf{a}$. Feature **a** corresponds to the OSGI support library, while **b** and **c** are two implementations of the component structure that rely on this library. On the visualization, we observe that nodes b and c are slightly taller than other nodes, and node **a** is wide. Using our tool support for the Variability

[4]http://smalltalkhub.com/#!/~abergel/Familiar
[5]http://familiar-project.github.io
[6]https://github.com/surli/spinefm

[7]In the following versions V2 and V3 are not detailed steps, but the intermediate metrics are shown to their evolution.

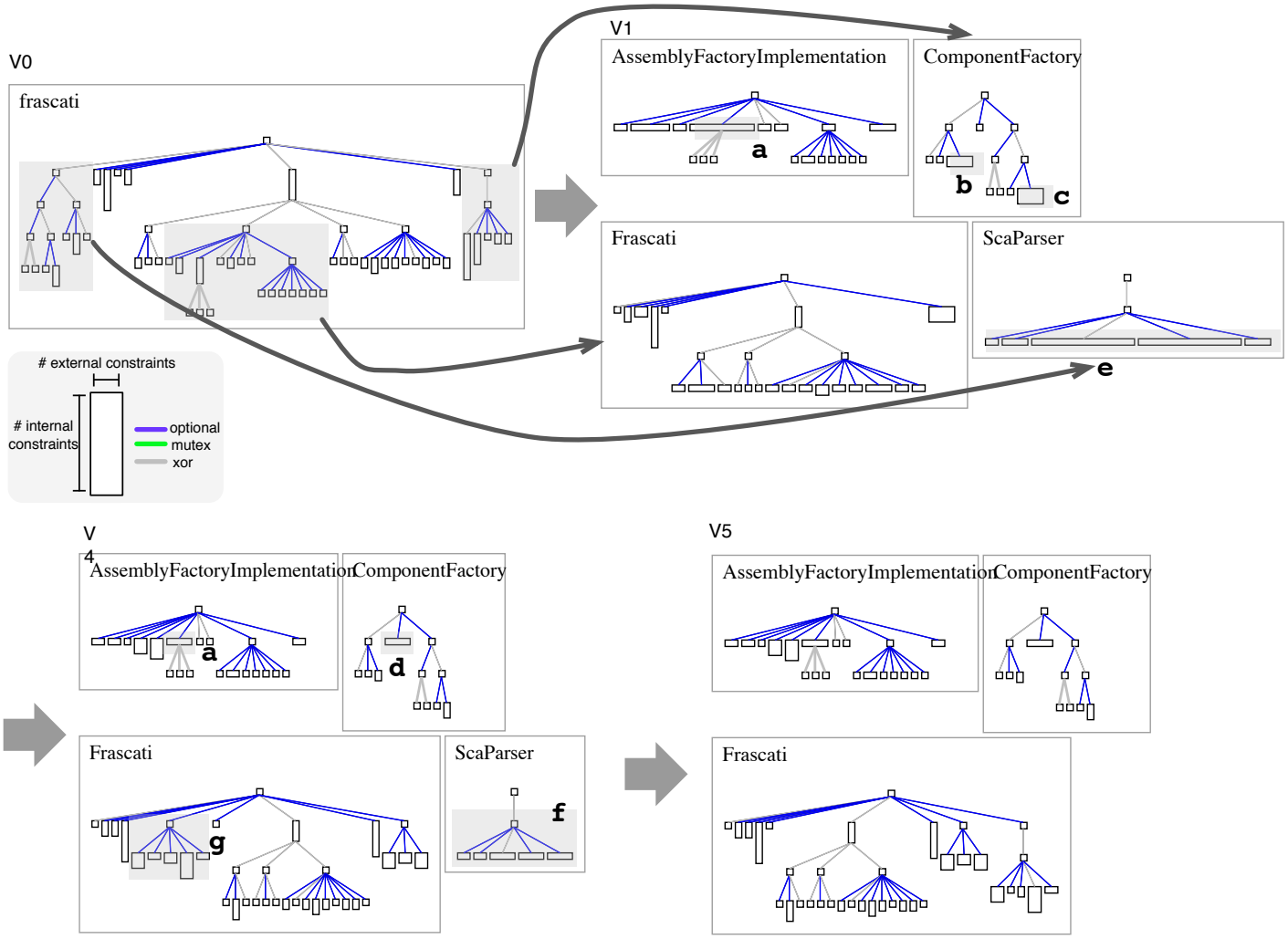

Fig. 5: Evolution of the FraSCAti decompositions

Blueprint, hovering the mouse pointer over these nodes gives the exact list of dependencies. Note that the width of node **a** is greater than the height of nodes **b** and **c**, meaning that node **a** contains additional incoming dependencies.

In V4, the dependencies from **b** to **a** and **c** to **a** have been replaced by 3 dependencies, $a \implies d, c \implies d$ and $d \implies a$, d acting as a proxy inside the `ComponentFactory` FM. The 2 inter-FMs dependencies of V1 have also been replaced by 2 intra FM constraints and 1 inter-FMs dependencies. As a result the modularity of V2 is strengthened thanks to the removal of one external dependency. The feature **a** has one external dependency less, therefore its width is reduced compared to V1.

In V1, a majority of features of `ScaParser` are involved in many external constraints to `Frascati` features. To avoid this coupling, a complex proxy to these features is introduced in the `Frascati` FM, consecutively reducing the number of external constraints in both FMs. In Figure 5, we observe that the whole set of nodes marked **g** in V4 have been introduced in the `Frascati` FM to act as proxies to the nodes marked **f** in `ScaParser` (*i.e.,* each child node in the hierarchy **f** is mapped into a child node in **g**). In table I, we see that the amount of external dependencies are reduced from V1 to V4 for `ScaParser`, as the exCC metric is moving from 17 to 8, 21 to 8 and 16 to 8 for the three more complex FMs. The exCTC is also reduced or kept equals (this is the case for `ScaParser` as the number of features involved in constraints is still the same among versions).

V4 → V5. In V5 the `ScaParser` FM has been removed and merged back inside the `Frascati` FM. Visualizing the dependencies with the main architect of FraSCAti, he realized that the hierarchy f in `ScaParser` is almost entirely mapped into the hierarchy g and that the separation of the parsing features is not offering much in the understanding of the whole platform variability. In V5 we observe that child nodes in `ScaParser` (at the bottom right part of the `Frascati` FM on Figure 5) remains wide because of the dependencies to the `AssemblyFactoryImplementation` FM.

TABLE II: Metrics about successive Frascati FMs (low values for CC, CTC, exCC, and exCTC are positive indicators)

Version	#FM	#features	CC	CTC	exCC	exCTC
V0	1	63	46	57%	0	0%
V1	4	64	4	16%	14.5	47%
V2	4	65	4	17%	14.5	47%
V3	4	67	6	28%	11	42%
V4	4	77	11	35%	6.5	34%
V5	3	71	13	45%	5	19%

Summary. Table II gives the averaged metrics for the consecutive versions. We first observe that the different steps V1 to V5 have increased the CTC (*i.e.,* decomposed FMs have more internal constraints and are likely to be more easily understandable and configurable). Both exCC and exCTC are relatively high in steps V1 to V4, but the final simplification in V5 shows on average 5 constraints per FM and a 19% exCTC, which is low for both metrics. As a result, the visualization blueprint has helped in understanding the constraints of the decomposed FMs and their impacts. Together with its interactive functionalities, it was possible for the SPL maintainer to try different decompositions that still fit the domain while providing better maintainability.

B. Visualizing Concerns of a Complex SPL

Our second case study concerns the YourCast SPL. YourCast[8] is a project aiming at creating an industrial-strength SPL for *Digital Signage Systems* (DSS) [13]. A DSS broadcasts dynamic information, mainly from the Web, typically targeting both public institutions and private companies. Over a two-year period, this project involved around 30 contributors carrying out more than 470 KLOC for the software assets of the SPL and two SPL architects.

[8]http://www.yourcast.fr

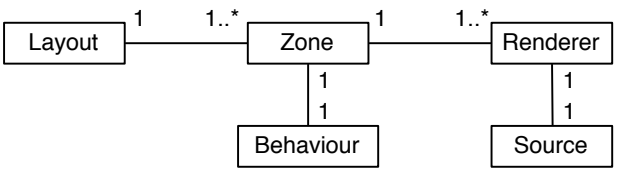

Fig. 6: Yourcast domain model

Separating FMs was a necessity to tame the complexity of the variability and to handle the frequent evolutions among features and constraints [30]. In this case study the decomposition of FMs is thus directly driven by a domain model. Each domain model instance has an associated FM that captures its variability. Figure 6 shows the domain model of YourCast SPL (*i.e.,* how the different concerns of a DSS are decomposed). Information comes from a source, a renderer displays it inside a zone, which has a specific behavior (*e.g.,* horizontal scrolling). A layout organizes zones on a screen.

Using Variability Blueprint, the objective in this case is first to have a high-level picture of the variability of the SPL. We then expect to analyze complex parts and understand whether they come from the domain or from an inadequate organization of the SPL concerns. Figure 7 shows Variability Blueprint of the YourCast SPL, while Table III shows the metrics corresponding to each FM.

On Figure 7, we observe that the blueprint is quite effective in showing different patterns, with very thin and tall nodes, but also *square* boxes as in the upper left FM. The blueprint uses interactive highlights of features. A green-colored feature is the feature pointed by the mouse cursor. Impact is visually represented using colors. For example, orange features are deselected internally by the selection of the features (using the inter feature constraints mentioned above), blue features are

TABLE I: Detailed metrics about successive FraSCAti decompositions

Version	FM Name	# features	CC	CTC	exCC	exCTC
V0	Frascati	63	46	57%	0	0
V1	Frascati	26	14	46%	17	54%
	ComponentFactory	12	2	17%	4	17%
	ScaParser	7	0	0%	21	71%
	AssemblyFactoryImplementation	19	0	0%	16	47%
V2	Frascati	26	14	46%	17	54%
	ComponentFactory	13	3	23%	4	15%
	ScaParser	7	0	0%	21	71%
	AssemblyFactoryImplementation	19	0	0%	16	47%
V3	Frascati	26	14	46%	17	54%
	ComponentFactory	13	3	23%	2	8%
	ScaParser	7	0	0%	16	71%
	AssemblyFactoryImplementation	21	7	43%	9	33%
V4	Frascati	36	30	72%	8	22%
	ComponentFactory	13	8	23%	2	8%
	ScaParser	7	0	0%	8	71%
	AssemblyFactoryImplementation	21	7	43%	8	33%
V5	Frascati	37	30	70%	6	16%
	ComponentFactory	13	3	23%	2	8%
	AssemblyFactoryImplementation	21	7	43%	8	33%

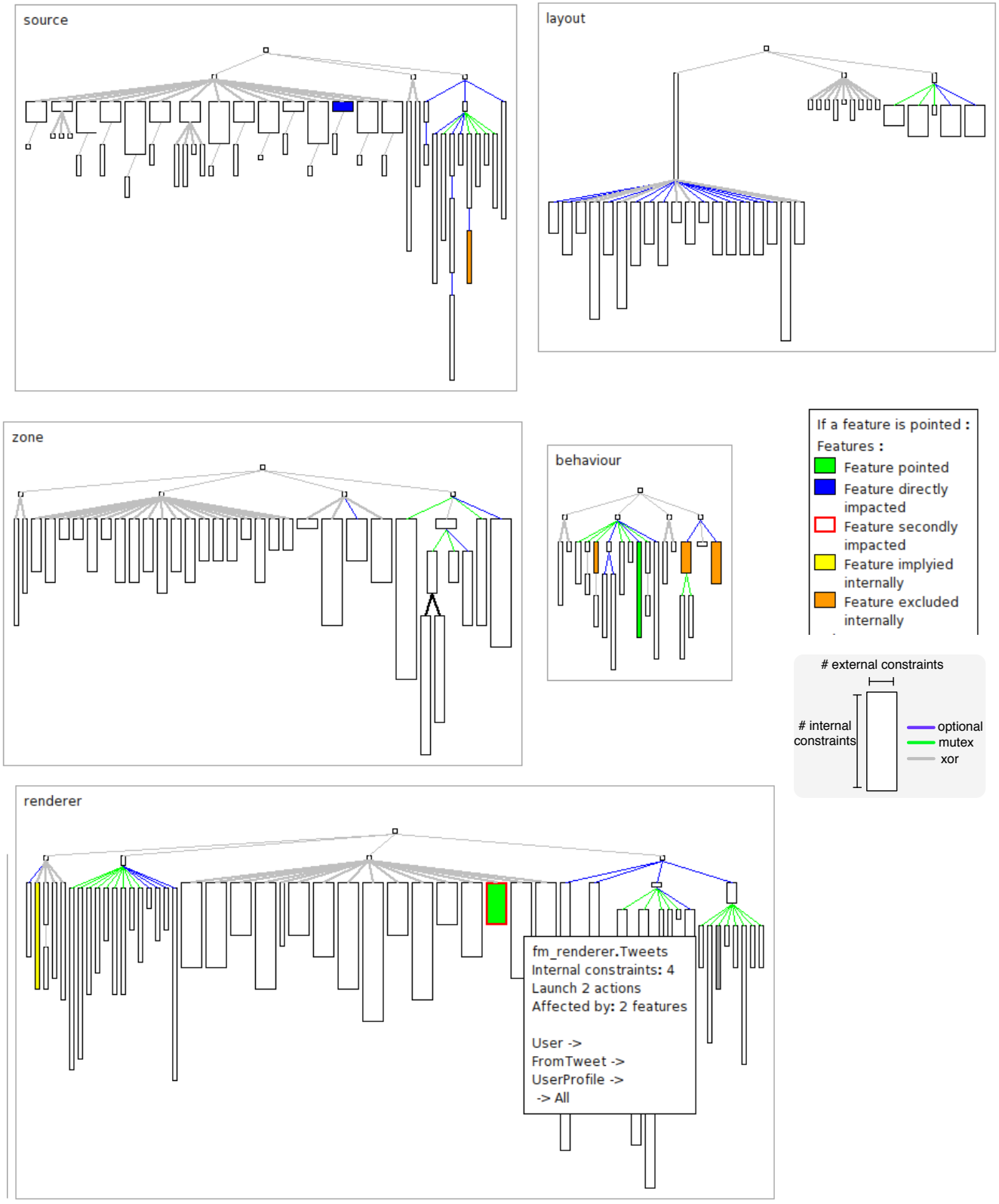

Fig. 7: Visualization of the Yourcast SPL

directly impacted by constraints.

TABLE III: Metrics about Yourcast FMs (low values for CC, CTC, exCC, and exCTC are positive indicators)

Concept	*#Features*	*CC*	*CTC*	*exCC*	*exCTC*
Sources	81	154	88%	20	18%
Renderers	76	347	93%	30	30%
Transitions	33	45	76%	5	6%
Zones	49	160	87%	52	65%
Layouts	51	59	95%	37	8%
Average	58	153	88%	28.8	26%

On Table III, we first observe that while the YourCast SPL may be considered a medium-size SPL with 290 features, it is highly complex with a great number of constraints (almost 900). Looking at the distribution of constraints, it can be seen that the large majority of them are internal constraints of the FMs, which is good for maintainability. Not surprisingly, with a lot of constraints in comparison to the number of features, the CTCs are very high with an average of 88%. This is also due to the specific way the FMs are built from the existing products [30].

Focusing on external constraints, the exCC and exCTC metrics are quite low for almost all FMs except `Zone`. The maintainability of the other FMs can be considered as good. As for `Zone`, we explain these high values by the fact that the `Zone` domain element is central in the model that organizes the decomposition. It is indeed the only element that is connected to 3 other elements, thus creating more external constraints on the FMs of these elements. To tame this remaining complexity, we could envisage the use of proxy features, as in the FraSCAti case study, to concentrate the external constrains on a smaller set of features.

V. Related Work

Visual support for FM configuration. As briefly discussed in Section II-A, the first visualization of feature models was introduced in the FODA method [2] with feature diagrams. From then on, the majority of proposed FM visualizations were targeted at facilitating the configuration process (selecting features in a FM to build a valid configuration). Usually SPL environments also rely on the feature diagram representation equipped with tick boxes to provide a configuration support.

Botterweck, Nestor et al. [14] were the first to introduce an interactive visualization during configuration (a feature diagram showing what is selected, unselected, deselect at each configuration step with some color-coding), relying on SAT-solving to update the visualization at each step.

Extending it, Botterweck, Thiel et al. [15] have proposed a meta-model and a visual support for configuration of FMs and associated components. Their model provides the notions of decision, feature, components and relationship between them, so that the impact between FM selection and components can be visualized. Their visualization notably supports details on demand and focus with zoom [16], but it is still based on a common feature diagram and cannot show several FMs at the same time. While we also rely on a metamodel, Variability Blueprint only focuses on feature models, but can handle relations between several of them. On the other hand, it does not handle relationships with other artifacts, but bridging our Variability Blueprint to SPL assets, such as components and their dependencies, is a promising perspective.

Another visualization tool for configuration is S2T2 [18], proposed by Lero[9]. S2T2 was designed to support the configuration of FMs while displaying efficiently constraints between features. Our visualization has a different focus since it aims at being a visual support for maintenance and reengineering.

Pleuss et al. [17] made an overview of possible visualization techniques (*e.g.,* tree maps, cone trees, Venn diagrams) for FMs with criteria to determine their benefits and drawbacks for interactive product configuration. The study focuses on several expected benefits, *e.g.,* showing the hierarchy, the constraint dependencies, a possible configuration workflow. Interestingly, their comparison matrix shows that none of the studied techniques is well suited to display both constraints dependencies and complex feature models. On the contrary we show in our two case studies that our Variability Blueprint is well adapted in these situations.

Understanding FMs. Some related work is also concerned by the comprehension of feature models. Jaksic et al. [23] conducted an experiment to compare textual and graphical representations during the edition and manipulation of FMs. Their results focus on the quality of the produced FMs (absence of errors) and show the advantages of the graphical representation, being common feature diagrams, to increase cognitive efficiency and effectiveness of novice practitioners.

Reinhartz-Berger et al. have studied the comprehensibility of the CVL standard for feature modeling [34] with some controlled experiments in which users have to answer questions related to the interpretation of a FM. They notably showed that some constructs (or-group) are perceived as more difficult, whatever the expertise level of users. The same authors have also studied CVL and OVM, two modeling languages supporting relations from feature models to other development artifacts, showing that the variability models are easily comprehensible whereas the relations have a low comprehensibility [35].

VI. Conclusion

Feature models (FMs) are widely used to compactly represent the variability of a given domain or system. To handle the complexity of large and complex FMs, an appropriate visual support should complement existing management techniques that support separation of concerns and decomposition into related FMs.

We presented a visual and interactive blueprint that enables a software product line (SPL) maintainer to review the complexity of these related FMs. The visual representation reuses the usual tree-like hierarchy of a feature diagram, removing names to benefit the contrasts between internal and external constraints on features of several FMs. Some interactions on the visual representation complement it with details on a specific feature.

[9]http://download.lero.ie/spl/s2t2/

A practitioner can then decompose and relate sub-domains or concerns of a SPL, while being able to visually validate that the complexity is tamed in the resulting FMs. The tool-supported approach is shown to be efficient in two different real case studies by appropriate metrics related to maintainability.

As future work, we plan to further evaluate the practicality of the proposed solution, as well as the coupling with automated decomposition techniques. Together we hope they contribute to a methodology of FM management for SPL practitioners.

ACKNOWLEDGEMENTS

We gratefully thank our sponsors. This work has been partially founded by the *ASPIRE: Assess, Visualize and Reengineer Software Engineering Processes* Ecos-Conicyt C11E05 project, by the ANR YourCast project under contract ANR-2011-EMMA-013-01. We also thank Renato Cerro for his feedback on an early version of the paper.

REFERENCES

[1] K. Pohl, G. Böckle, F. J. van der Linden, Software Product Line Engineering: Foundations, Principles and Techniques, Springer-Verlag, 2005.

[2] K. Kang, S. Cohen, J. Hess, W. Novak, S. Peterson, Feature-oriented domain analysis (FODA) feasibility study, Tech. Rep. CMU/SEI-90-TR-21, SEI, CMU (November 1990).

[3] P.-Y. Schobbens, P. Heymans, J.-C. Trigaux, Y. Bontemps, Generic semantics of feature diagrams, Computer Networks 51 (2) (2007) 456–479.

[4] K. Czarnecki, A. Wasowski, Feature diagrams and logics: There and back again, in: 11th International Software Product Line Conference (SPLC'07), IEEE, 2007, pp. 23–34.

[5] D. Benavides, S. Segura, A. Ruiz-Cortes, Automated analysis of feature models 20 years later: a literature review, Information Systems 35 (6).

[6] T. Berger, R. Rublack, D. Nair, J. M. Atlee, M. Becker, K. Czarnecki, A. Wasowski, A survey of variability modeling in industrial practice, in: Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems, VaMoS '13, pp. 7:1–7:8.

[7] M. Acher, P. Collet, P. Lahire, R. France, Separation of Concerns in Feature Modeling: Support and Applications, in: Aspect-Oriented Software Development(AOSD'12), ACM, 2012.

[8] M. Acher, P. Collet, P. Lahire, R. B. France, Slicing feature models, in: Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering, ASE '11, IEEE Computer Society, Washington, DC, USA, 2011, pp. 424–427.

[9] A. Metzger, K. Pohl, P. Heymans, P.-Y. Schobbens, G. Saval, Disambiguating the documentation of variability in software product lines: A separation of concerns, formalization and automated analysis, in: RE'07, 2007, pp. 243–253.

[10] D. Dhungana, P. Grünbacher, R. Rabiser, T. Neumayer, Structuring the modeling space and supporting evolution in software product line engineering, Journal of Systems and Software 83 (7) (2010) 1108–1122.

[11] D. Dhungana, D. Seichter, G. Botterweck, R. Rabiser, P. Gruenbacher, D. Benavides, J. A. Galindo, Configuration of multi product lines by bridging heterogeneous variability modeling approaches, in: SPLC'11, IEEE, 2011, pp. 120–129.

[12] A. Hubaux, P. Heymans, P. Y. Schobbens, D. Deridder, E. K. Abbasi, Supporting multiple perspectives in feature-based configuration, Software and Systems Modeling 12 (2013) 641–663.

[13] S. Urli, M. Blay-Fornarino, P. Collet, S. Mosser, M. Riveill, Managing a Software Ecosystem Using a Multiple Software Product Line: a Case Study on Digital Signage Systems, in: Euromicro Conference series on Software Engineering and Advanced Applications(SEAA'14), special issue: Software Product Lines and Software Ecosystems, , Elsevier, Verona, Italy, 2014, pp. 1–8.

[14] G. Botterweck, D. Nestor, A. Preußner, C. Cawley, S. Thiel, Towards supporting feature configuration by interactive visualisation, in: Software Product Lines, 11th International Conference, SPLC 2007, Kyoto, Japan, September 10-14, 2007, Proceedings. Second Volume (Workshops), Kindai Kagaku Sha Co. Ltd., Tokyo, Japan, 2007, pp. 125–131.

[15] G. Botterweck, S. Thiel, D. Nestor, S. bin Abid, C. Cawley, Visual tool support for configuring and understanding software product lines, in: Software Product Line Conference, 2008. SPLC '08. 12th International, 2008, pp. 77–86.

[16] D. Nestor, S. Thiel, G. Botterweck, C. Cawley, P. Healy, Applying visualisation techniques in software product lines, in: Proceedings of the 4th ACM symposium on Software visualization, SoftVis '08, ACM, New York, NY, USA, 2008, pp. 175–184.

[17] A. Pleuss, R. Rabiser, G. Botterweck, Visualization techniques for application in interactive product configuration, in: Proceedings of the 15th International Software Product Line Conference, Volume 2, SPLC '11, ACM, New York, NY, USA, 2011, pp. 22:1–22:8.

[18] A. Pleuss, G. Botterweck, Visualization of variability and configuration options, STTT 14 (5) (2012) 497–510.

[19] M. Lanza, S. Ducasse, Polymetric views—a lightweight visual approach to reverse engineering, Transactions on Software Engineering (TSE) 29 (9) (2003) 782–795.

[20] C. Anslow, S. Marshall, J. Noble, E. Tempero, R. Biddle, User evaluation of polymetric views using a large visualization wall, in: Proceedings of the 5th International Symposium on Software Visualization, SOFTVIS '10, ACM, New York, NY, USA, 2010, pp. 25–34.

[21] R. Wettel, M. Lanza, R. Robbes, Software systems as cities: a controlled experiment, in: Proceedings of the 33rd International Conference on Software Engineering, ICSE '11, ACM, New York, NY, USA, 2011, pp. 551–560.

[22] S. Mosser, A. Bergel, M. Blay-Fornarino, Visualizing and assessing a compositional approach of business process design, in: Proceedings of 9th International Conference on Software Composition (SC'10), LNCS Springer Verlag, 2010, to appear.

[23] A. Jakšić, R. B. France, P. Collet, S. Ghosh, Evaluating the usability of a visual feature modeling notation, in: Software Language Engineering, Springer, 2014, pp. 122–140.

[24] T. Thüm, D. Batory, C. Kästner, Reasoning about edits to feature models, in: 31st International Conference on Software Engineering (ICSE'09), ACM, 2009, pp. 254–264.

[25] E. Bagheri, D. Gasevic, Assessing the maintainability of software product line feature models using structural metrics, Software Quality Control 19 (3) (2011) 579–612.

[26] M. Mendonca, A. Wasowski, K. Czarnecki, Sat-based analysis of feature models is easy, in: Proceedings of the 13th International Software Product Line Conference, SPLC '09, Carnegie Mellon University, Pittsburgh, PA, USA, 2009, pp. 231–240.

[27] M.-O. Reiser, M. Weber, Multi-level feature trees: A pragmatic approach to managing highly complex product families, Requir. Eng. 12 (2) (2007) 57–75.

[28] A. Hubaux, M. Acher, P. Heymans, P. Collet, P. Lahire, Separating Concerns in Feature Models: Retrospective and Support for Multi-Views, , Springer, 2013, pp. 3–28.

[29] H. Hartmann, T. Trew, Using feature diagrams with context variability to model multiple product lines for software supply chains, in: SPLC'08, IEEE, 2008, pp. 12–21.

[30] S. Urli, M. Blay-Fornarino, P. Collet, Handling complex configurations in software product lines: A tooled approach, in: Proceedings of the 18th International Software Product Line Conference - Volume 1, SPLC '14, ACM, New York, NY, USA, 2014, pp. 112–121.

[31] L. Seinturier, P. Merle, D. Fournier, N. Dolet, V. Schiavoni, J.-B. Stefani, Reconfigurable SCA applications with the FraSCAti platform, in: SCC'09: Proceedings of the 6th International Conference on Service Computing, Bangalore, India, 2009, pp. 268–275.

[32] M. Acher, A. Cleve, P. Collet, P. Merle, L. Duchien, P. Lahire, Extraction and evolution of architectural variability models in plugin-based systems, Softw. Syst. Model. 13 (4) (2014) 1367–1394.

[33] J. Marino, M. Rowley, Understanding SCA (Service Component Architecture), 1st Edition, Addison-Wesley Professional, 2009.

[34] I. Reinhartz-Berger, K. Figl, Ø. Haugen, Comprehending feature models expressed in cvl, in: Model-Driven Engineering Languages and Systems, Springer, 2014, pp. 501–517.

[35] I. Reinhartz-Berger, K. Figl, Comprehensibility of orthogonal variability modeling languages: The cases of cvl and ovm, in: Proceedings of the 18th International Software Product Line Conference - Volume 1, SPLC '14, ACM, New York, NY, USA, 2014, pp. 42–51.