Textual Variability Modeling Languages

An Overview and Considerations

Maurice H. ter Beek
ISTI–CNR, Pisa, Italy
maurice.terbeek@isti.cnr.it

Klaus Schmid
University of Hildesheim, Germany
schmid@sse.uni-hildesheim.de

Holger Eichelberger
University of Hildesheim, Germany
eichelberger@sse.uni-hildesheim.de

ABSTRACT

During the three decades since the invention of the first variability modeling approach [28], there have been multiple attempts to introduce advanced variability modeling capabilities. More recently, we have seen increased attention on textual variability modeling languages. In this paper, we summarize the main capabilities of state of the art textual variability modeling languages, based on [23], including updates regarding more recent work. Based on this integrated characterization, we provide a discussion of additional concerns, opportunities and challenges that are relevant for designing future (textual) variability modeling languages. The paper also summarizes relevant contributions by the authors as input to further discussions on future (textual) variability modeling languages.

CCS CONCEPTS

• **Software and its engineering → Specification languages**; **Software product lines**.

KEYWORDS

software product lines, variability modeling, textual specification languages

ACM Reference Format:
Maurice H. ter Beek, Klaus Schmid, and Holger Eichelberger. 2019. Textual Variability Modeling Languages: An Overview and Considerations. In *23rd International Systems and Software Product Line Conference - Volume B (SPLC '19), September 9–13, 2019, Paris, France.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3307630.3342398

1 INTRODUCTION

Since the very early days, variability modeling has mostly focused on graphical modeling [28], especially using feature diagrams in the form of trees. This has led to different notations, which often only varied in minor technical details [41]. Variability modeling was handled in numerous ways in practice using textual notations. Classical examples of these are KConfig [29] and CDL [43], textual variability description languages that have been created in the open source world. However, these kinds of languages typically suffer from the problem that they are not formally defined and very hard to analyze [25].

SPLC '19, September 9–13, 2019, Paris, France

ACM ISBN 978-1-4503-6668-7/19/09…$15.00
https://doi.org/10.1145/3307630.3342398

Textual variability modeling approaches were also invented in academia. Most follow the notion of feature modeling [41] with variations. Some also took the approach of decision modeling [40].

The goal of this paper is to summarize and update the categorization of textual variability modeling languages provided in [23] as a basis for discussing future options and challenges towards the design of a simple (textual) variability modeling language that the community can agree on. Due to space restrictions, we can, of course, not replicate that earlier survey. Thus, we also refer the reader to this earlier publication for further details [23].

2 A SHORT STATE OF THE ART OF TEXTUAL VARIABILITY MODELING LANGUAGES

In this section, we provide a summary of the overview on textual variability modeling languages presented in [23], updated with more recent approaches. This comprehensive review of existing textual variability modeling languages was based on both an analysis of the existing literature as well as contact with the corresponding authors to include potential feedback in order to ensure that the categorizations of the various languages was adequate. Thus, there may be (implemented) language capabilities that are not mentioned in the underlying literature or indicated by the involved authors and, therefore, not listed in this paper.

2.1 Updated Literature Analysis

The original analysis (cf. [23, Sect. 4]) considered these languages:

- *Feature Description Language (FDL)* [42] mainly aims at being a textual representation of feature diagrams.
- *Forfamel* [5] is part of the Kumbang approach; Forfamel aims at feature modeling, while Koalish adds structural modeling.
- *Tree grammars for representing cardinality-based feature models were introduced by Batory in [6].*
- *Variability Specification Language (VSL)* [1] integrates feature modeling with configuration links and variable entities.
- *Simple XML Feature Model (SXFM)* [32] is an XML-based representation of feature models.
- *FAMILIAR* [2], next to modeling variability, also includes capabilities for combining and analyzing variability models.
- *Text-based Variability Language (TVL)* [17] supports textual feature modeling, including capabilities for feature attributes, cardinalities and modularization.
- *μTVL* [16] is a variation of TVL, dropping some concepts, but also adding others like multiple trees in a single model.
- *CLAss, FEature, Reference approach (Clafer)* [13] combines meta-modeling of classes with feature modeling support.
- *VELVET* [35] is a language, inspired by TVL, but extends it in several directions and reimplements it from scratch.

- *INDENICA Variability Modeling Language (IVML)* [39] follows the decision modeling paradigm with a strong focus on ease of learnability, expressiveness, and scalability.

In the meantime, some additional approaches have been published:

- *Clafer (extended with behavior)* [27] extends [13] with a temporal dimension, resulting in a language combining behavior, structure, and variability.
- *PyFML* [3] is a textual feature modeling language based on the Python programming language.
- *Variability Modeling (VM)* [4] is a language that was developed in an industrial project, with specific constraints to ease reasoning particularly for applications in the video domain.

Although we are aware of more recent XML-based approaches than SXFM, such as, e.g., [45], we do not include them here, in particular if they do not add further capabilities or aim at a pure XML representation (on instance or schema level) of existing capabilities. The main focus of [23] was on classifying and summarizing the characteristics that are actually supported by the languages to better understand their peculiarities. Discussions of potential future capabilities as well as secondary characteristics like analyzability were excluded. We will discuss these aspects in Section 3. Thus, the subset of characteristics we consider in this section as a basis for our discussion in this paper is based on the following dimensions:

- Configurable elements
- Constraint support
- Configuration support
- Scalability support
- Language characteristics

In Table 1, we show a selection of the extensive classification of textual variability modeling languages from [23], updated for Clafer (extended with behavior), PyFML, and VM. From the above dimensions, we focus here on particular sub-dimensions (stated below between parentheses): configurable elements (forms of variation, attached information, cardinalities, references, and additional data types — such as basic types predefined by the language, user-defined types, or types derived from already known types), constraint support (constraint expressions), configuration support (default values, value assignment, and partial or complete configurations), scalability support (here only through composition, i.e., the capability of integrating units of configurable elements into a single model). Finally, we briefly discuss language characteristics, as reported in Table 7 in [23], in Section 3. As an additional dimension, we also report whether the language has a formal semantics.

We refer the interested reader to [23] for detailed pointers (including page numbers) to the literature that confirm the level of support offered by the surveyed textual variability languages. In Table 1, we simplify the notation to direct (+), indirect (±), unclear (?) or no (−) support. In the next sections, we summarize the classification, providing information on the type of attached information, cardinalities, and references that is supported, as well as details of the supported data types and to what they apply.

Compared with [23], we note the following updates. Clafer (extended with behavior) provides direct support for *simple* cross-tree constraints (cf. [27, p7]) and for so-called parallel decomposition of non-exclusive clafers, i.e., the selection of *multiple* features out of several possible variations (cf. [27, p13]).

PyFML provides direct support for *optional*, *alternative*, and *multiple* feature selection, for *attached information* in the form of attributes of *predefined* types (Boolean, Integer, Float, String), *simple* cross-tree constraints, constraint expressions in *propositional* logic as well as both *relational* and *arithmetic* constraint expressions, and configuration support by allowing *default values* and *value assignment* [3, p46].

As can be concluded from [4, Sect. 5], VM provides similar capabilities as PyFML, but also supports the specification of *cardinalities*, a modularization mechanism supporting *composition*, direct support for *partial configurations* and indirect support for *complete configurations*. In addition (not detailed in Table 1), VM supports constraint resolution hints such as delta values or objective functions.

From [23], we know that TVL and μTVL do not provide direct support for full-fledged *composition*, but merely for inclusion and conjunction. Finally, we are aware of *formal semantics* for FDL [42, p4ff], Forfamel (indirectly, by translation to WCRL) [5, p36], TVL [17, p1136ff], μTVL [16, p208ff], and Clafer [27, p2:23ff].

2.2 Configurable Elements

The basic elements of configuration in nearly all languages is a *feature*, respectively a *feature group*, the only exceptions are IVML, which uses decision variables, and Clafer, where it is a conceptual mix of structural and variability modeling, called a *clafer*.

All languages support *optional* and *alternative* variability, most do also support *multiple* selection, i.e., selecting at least two out of a range of possible features.

Most languages also cater for *attached information* to the basic variability unit. Mostly these are feature *attributes*. Sometimes this can also be *parameters* (VSL) or other *features* (Clafer). IVML also supports *meta-attributes*, which can, for example, express binding times, implementation advices, etc.

Most languages also support *cardinalities*. Typically, these are *feature* and *group cardinalities*. However, Tree grammars only support *feature cardinalities*, while SXFM and μTVL only support *group cardinalities*. Depending on the language design, *cardinalities* are typically realized as a special capability. In some cases, however, the language supports generic multiplicity and the details of *cardinalities* are expressed as constraints.

Many languages do also support the notion of (configuration) *references*, which simply alias other configurable elements. As most languages are very restricted with respect to what a configurable element can be, typically *references* are only possible to a single type. However, IVML differs insofar as it supports *references* to arbitrary *data types*. This enables much richer static (type) checking. As a consequence, IVML, as shown in [22], can also be used to model topological variability. Topological variability targets variations of connecting components with respect to a certain order, in specific interconnected hierarchies, in different quantities [11], or, in general, in terms of graph-like structures.

Related to the question of the basic elements for expressing variability is the question to what extent *data types* are supported. In most cases, there is a basic *feature* (or *clafer*) data type. The only exception is IVML, which can combine the basic element of variation with all available *data types*.

Table 1: Language support for configurable elements, type systems, constraints, configurations, scalability, and semantics

	forms of variation							data types			constraint expressions					configurations					
Language	optional	alternative	multiple	extension	attached info	cardinalities	references	predefined	derived	user-defined	simple	propositional	first-order	relational	arithmetic	default values	assign values	partial	complete	composition	formal semantics
FDL	+	+	+	−	−	±	−	−	−	−	+	−	−	−	−	+	−	−	−	−	+
Forfamel	+	+	+	+	+	+	+	−	−	+	−	+	+	+	+	−	+	−	+	−	±
Tree grammars	+	+	+	−	−	+	−	−	−	−	−	+	−	−	−	−	−	−	−	−	−
VSL	+	+	+	+	+	+	+	+	−	+	+	?	?	−	?	+	+	+	+	+	−
SXFM	+	+	+	−	−	+	−	−	−	−	−	?	−	−	−	−	−	−	−	−	−
FAMILIAR	+	+	+	−	−	−	?	+	+	−	−	+	−	−	−	−	+	+	+	+	−
TVL	+	+	+	?	+	+	+	+	−	+	−	+	−	+	+	−	+	−	−	−	+
μTVL	+	+	±	+	+	+	−	+	−	−	+	+	−	+	+	−	?	+	+	−	+
Clafer	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	−	+	+	+	+	+
VELVET	±	+	+	+	+	±	+	+	−	−	−	+	−	+	−	+	+	+	±	+	−
IVML	+	+	+	+	+	±	+	+	+	+	−	+	+	+	+	+	+	+	±	+	−
PyFML	+	+	+	−	+	−	−	+	−	−	+	+	−	+	+	+	+	−	−	−	−
VM	+	+	+	−	+	+	−	+	−	−	+	+	−	+	+	+	+	+	±	+	−

Some languages allow the derivation of new types, e.g., through a form of inheritance like in object-oriented languages, type composition, container types, or even composing types with constraints, leading to *type restrictions*. However, in most languages these further *data types* are restricted to feature attributes.

2.3 Constraint Support

The various languages differ considerably in terms of their capabilities for *constraint expressions*. Overall, several layers of expressiveness can be distinguished. As a general rule, these are ordered in increasing levels of expressiveness. However, this correlates also to a decreasing level of analyzability (cf. Section 3).

Especially in diagrammatic presentations often basic requires and excludes relationships are present *(simple dependencies)*. However, in textual languages these are rarely to be found. Rather all languages, except for FDL, at least support full *propositional logic*.

As *simple dependencies* can be seen as special cases of propositional logic, there is no need to have both. However, four languages combine *simple dependencies* and *propositional logic*, probably using *simple dependencies* as shortcuts for otherwise more complex *propositional logic* or just to be close to classical feature modeling publications. *Propositional logic* can be extended, e.g., by supporting *relational expressions* or *arithmetic expressions*.

Some languages also support quantification over formulas, which enables for example to give constraints over all subtrees. In [23], this is (not fully correctly) called *first-order logic*.[1] While this is a very powerful construct, it is only available in four languages considered in this survey: Forfamel, VSL, Clafer, and IVML.

[1]Most languages support the quantifiers, but not necessarily the predicates, functions and constants typical of first-order logic.

A mechanism, which is only available in IVML, is the use of default constraints. These are constraints that can be altered as part of the constraint-resolution process. In particular, together with scoped imports (cf. Section 2.5) this leads to (restricted) support for non-monotonic reasoning.

2.4 Configuration Support

While graphical variability modeling notations are typically focused only on the modeling, it is actually rather common for textual variability modeling languages to support the *configuration* as well. The most elementary category is the *value assignment*, which is supported by all languages but FDL, Tree grammars, and SXFM.

Some languages (e.g., FDL, VSL, VELVET, IVML, PyFML, and VM) also support *default values*, i.e., values that can be overridden at a later stage. In particular, VM differentiates between static and runtime configurations, using runtime tags as annotation to indicate the *binding time* of features and attributes, allowing the code to increase or decrease values at runtime.

Besides these basic capabilities of setting values, many languages also support the notion of a *configuration* as a first-class concept. In particular all languages except FDL, Forfamel, SXFM, and PyFML. These languages allow to designate a range of *value assignments* as a configuration and manage it separately. In most cases this can also be a *partial configuration*. However, not all approaches do support full separation in the sense that arbitrary many configurations can be separately managed from the basic model description.

2.5 Scalability Support

The earliest languages, namely FDL, Forfamel, Tree grammars, and SXFM, do not provide mechanisms for large-scale variability

modeling through *composition*, and neither does one of the most recent ones, PyFML. TVL and μTVL only allow the *inclusion* and *conjunction*, respectively, of models.

The remaining languages all support some form of scalability through *composition*. Clafer and VELVET do so by *inheritence*, whereas FAMILIAR provides two explicit *composition* operators, the 'merge' operator for overlapping and the 'aggregate' operator for disjoint models. VM supports the *import* of so-called (model) packages. IVML supports the *scoped import* of models, which besides the provisioning of namespaces also provides scopes for the reasoning process. It also provides an interface concept.

2.6 Language Characteristics

While fundamentally all languages described here are textual, they conceptually differ significantly. This is related to where they originate from and to the major sources of inspiration they rely on.

Some, especially the early languages, followed the idea of a tree-like feature diagram rather faithfully and focused on providing a corresponding syntax. Other languages used programming languages like C and Java (VSL, TVL, and IVML) or Python (PyFML) as an inspiration for their approach to syntax. There have also been proposals that rely on XML (e.g., SXFM) and languages like OCL (for IVML) and Alloy have been sources of inspirations, too. Finally, VELVET and μTVL are special cases as they themselves rely on another variability language TVL (which is inspired by C). One of the rationales for using programming languages as a basis is to make it easier and more natural for users to apply these languages.

As predominant structures of the languages we basically find only three alternatives. Some are tree-based, i.e., a tree structure is replicated textually, while others are graph-based ones, i.e., focusing on representing textually a graph structure. Finally, some languages are driven by (potentially nested) declarations of variables, which can then be used along with value assignments to represent, for example, tree as well as graph structures.

3 OTHER CONCERNS IN LANGUAGE DESIGN

In this section, we discuss further aspects that we believe to be relevant for making good choices in language design for future (textual) variability modeling languages. We focus here on seven topics, namely first citizen concepts, quantitative variability modeling, ecosystem support, 'exotic' modeling capabilities, binding time, analyzability, and modular language design.

3.1 First Citizen Concept

Almost all the approaches discussed in this paper use some form of feature as their main language concept. Notable exceptions are Clafer, which is based on an amalgamation of classes and features, and IVML, which represents variability decisions in terms of typed variables, i.e., follows the decision modeling paradigm [40]. However, one should also take into account that there strong relationships among the different paradigms, not only on a conceptual level [18], but in some cases even a formal correspondence has been shown [24].

In fact, features in their different notions are still considered as the predominant variability modeling approach in both industry [12] and academia [34]. However, according to [34], several aspects largely remain unexplored, e.g., non-functional (quantitative) properties, or merely exist as research topics that are currently not sufficiently taken up by industry, e.g., software ecosystems, multi-product lines, or dynamic software product lines.

This is in particular an issue, as without industrial cases, more recent topics remain academic ideas not really contributing to the evolution of variability modeling approaches. Moreover, approach and tool qualities like usability or scalability are typically only illustrated in terms of examples or not studied at all [34]. The authors believe that it is time to consider, explore, evaluate, and experiment with alternative first-level modeling concepts that integrate beneficial aspects of feature-based approaches, e.g., hierarchy and decomposition, with perceived advantages of textual variability modeling, e.g., scalability, as well as currently less explored needs of actual and future real-world variability modeling.

3.2 Quantitative Variability Modeling

Recently, there is growing interest in variability modeling (and analysis) techniques that explicitly consider quantitative aspects, which are particularly relevant to non-functional requirements, such as dependability, energy consumption, security, and cost. Since today's software is often embedded in smart and critical systems that run in environments where events affecting the system occur randomly, quantitative variability modeling is currently a hot topic.

This is reflected by the recent panel at VaMoS'19, which addressed questions like "How to incorporate quantities in (textual) languages for variability modeling?" [7], and the forthcoming special issue on quantitative variability modeling and analysis [8]. In [9], a rich, high-level textual DSL for configurable software-intensive systems was defined, with variability defined in terms of features and offering advanced quantitative constraint modeling options. The approach comes with tool support [10] and it can cope with the complexity of (re)configurable systems stemming from variability, behavior, and randomness. A related approach is provided by IVML [23].

3.3 Ecosystem Support

Some languages claim explicit modeling support for variations in software ecosystems. Bosch postulates that ecosystems are a natural extension of classical product lines [14] insofar as extending the notion of variability to open systems. Indeed, analysis of existing software ecosystems like Eclipse or the Linux package system show that open forms of variability description have evolved independently to support both open, distributed development as well as variability management [36]. In contrast to traditional (closed) variability modeling, open variability allows for the extension of the configuration space by variabilities that are not part of the core product line. In particular, an extension needs to be possible to 3rd parties who are not able to change the initial model. Thus, open variability also requires distributed modeling. Further, this requires particular capabilities of variability languages, as discussed in [37], like modularization and hiding of variabilities. While most other variability modeling approaches restrict themselves to 'closed' variability, IVML aims to support also open ecosystems.

Besides the aforementioned requirements, IVML supports defaults and non-monotonic reasoning capabilities, which have been

derived as being necessary from industrial cases in [15]. EASy-Producer, of which IVML is one component, also aims to support concepts like the feature pack approach, which aims to modularize feature groups along with their implementation in software ecosystems [30].

3.4 'Exotic' Modeling Capabilities

Exploring actual and future needs, besides those already discussed above, may require capabilities, which are currently not (well) supported by (textual) variability management approaches. Some examples are topological configurations with related constraints [11], behavioral aspects [31], or the specification of configuration optimization goals [4, 13]. Some proposals were made, e.g., how to model topological variability including constraints and reasoning in an integrated manner [22], or how to specify behavioral aspects through temporal constraints [27]. However, we perceive a certain reservation of the community against such 'exotic' capabilities, which is in contrast to indications of respective practical and industrial needs, such as in [11].

3.5 Binding Time

An important concern in software product line engineering is not only the specific configuration that is defined, but also when the configuration is determined, respectively applied to the system. One should note that these are ultimately two different notions. For example, if a configuration tool is needed for determining the configuration, then this is typically happening during the development process. On the other hand, the value may only have an effect very late in the process, e.g., a configuration may be instantiated when the system starts, a configuration file is read, and programmatic binding (e.g., through class-loading) happens.

Often both notions are referred to by the term *binding time*, although they are notably different. We propose the terms *definition time* and *binding time* in order to differentiate between them. Thus, the term definition time could be used to refer to something like the stages introduced by Czarnecki et al. [19]. Regarding binding time, Dolstra et al. [20] point out that this is not necessarily well-defined as there may be multiple points in time when an instantiation may happen, even for the same variability. They call this timeline variability. We can regard both definition time and binding time as descriptive information about a variability, which can be variable itself. Beyond those two, other aspects like different implementation technologies and so forth may be relevant, too. Hence, these may need to be represented as well. In [38], the term *meta-variability* was proposed for this more generic concept. Along with this, the authors proposed a very generic implementation approach using aspect-oriented programming for timeline variability.

Existing textual variability languages typically do not address the notion of binding time — and even those which do would hardly be able to represent the richness of concepts outlined above, because they restrict themselves to a single category with predefined unique values. They are thus not able to represent both definition time and binding time and are also not able to capture multiple alternatives which are needed for representing timeline variability. Notable exceptions are VM, as anticipated in Section 2.4, and in particular IVML. IVML directly implements the notion of meta-variability by allowing to attach arbitrarily many meta-decisions to any variability, which can also be set-typed to support timeline variability. As both values and meta-variabilities are user-definable, a context-specific binding time granularity as well as issues like technology variabilities can be supported on a per-need basis.

3.6 Analyzability

This is often a major concern as static analysis of variability models and product lines as a whole can provide very valuable assistance both during modeling and in derivation of product lines [44]. Unfortunately, there is an important trade-off between analyzability and expressiveness, as the more expressive a variability language is, the harder it is to analyze it. While propositional language is decidable and very efficient provers exist to handle it, analysis of higher levels of logic is not only significantly less efficient, it may often even be undecidable. On the other hand, the expressiveness of a higher-level logic may make certain things much easier to express, if not be a precondition for being able to represent the situation adequately in the first place.

Based on this observation some of the authors categorized different levels of expressiveness vs. analyzability trade-offs in earlier work in order to create a map of the situation [21], where four main classes could be identified: *basic variability modeling*, basically corresponding to classic feature models and mappable to propositional logic. *Cardinality-based variability modeling* gets particularly complex if potentially unbounded cardinalities are allowed. This leaves then the realm of decidability. Then *non-Boolean variability modeling* brings its own problems in terms of analyzability, e.g., the need for supporting arithmetic theories. Finally, *configuration references* can lead to significant challenges as they allow for arbitrary aliasing. However, this can also be used to great benefit, e.g., in the context of topological modeling [22].

Due to the inherent trade-off involved, any decision regarding the expressiveness of a language should be made carefully. The identification of different classes of expresssiveness also encourages the definition of different language levels within a modular language design as we will discuss below. Each of these could then have different reasoning support. However, the situation is not as simple as it may seem. As has been discussed by Eichelberger et al. [21, Table 1], many other aspects, like whether quantifiers are supported in constraints, also have a significant impact on analyzability and expressiveness. Making all of these aspects simultaneously customizable may create significant complexity.

3.7 Extensible Language Design

Most languages discussed here focus on a complete solution for variability modeling, while the contribution sometimes concentrates on differences, improvements, or additions over an existing language. For example, μTVL and VELVET extend TVL, Clafer was extended by temporal constraints, and VM provides domain-specific improvements over FAMILIAR. Such extensions do not always require a completely new language. One alternative could be an extensible language design, e.g., capabilities of extending a given language or embedding concepts into a host language.

Similar concepts are known from domain-specific languages (DSLs) [26], where external DSLs are complete languages for a certain purpose, while internal or embedded DSLs utilize and extend the concepts of a host language. As many textual variability languages are realized in terms of DSLs using related tooling, designing a variability modeling language for extensibility may support the experimentation and development of new approaches based on existing concepts and implementations.

3.8 Modular Language Design

In addition to the idea of extending a variability modeling language, conceptual differences may also be realized in terms of different *language levels*. In addition to pure language management (and product lines of variability modeling languages), this would allow to explicitly face trade-offs in the language design rather than aiming for a single general-purpose variability modeling approach.

Similar situations exist if languages explicitly combine basic and advanced modeling concepts (e.g., IVML and VM) or provide increasing capabilities of the same language concept, e.g., constraint capabilities (cf. Forfamel, Clafer, or IVML, as indicated in Table 1). In particular, for constraints and some specific modeling concepts, various trade-offs between expressiveness and analyzability do exist, as discussed in [21]. While simple forms of constraints such as pure Boolean expressions can be efficiently analyzed and solved [33], more complex constraints such as quantorized or temporal constraints are not decidable anymore.

In such settings, we can imagine that a *modular language design* could enable the product line engineer to focus on the most appropriate language level(s) for the situation at hand. Moreover, using just the needed language modules may allow for an automated selection of the most appropriate reasoning or analysis mechanisms, in turn leading to better performance or supporting analysis capabilities that are not available for the full language. Ultimately, a modular language design can foster the reuse of language levels (including the underlying language infrastructure), support the development of domain specific modeling capabilities (as suggested in [4]), and ease experiments as well as prototyping and the development of new variability modeling concepts.

4 CONCLUSION

We have presented an overview of the main characteristics of thirteen textual variability modeling languages, based on a systematic literature analysis reported in [23]. Beyond the coverage of this survey, we have incorporated two recently introduced languages (PyFML and VM) and updated the knowledge about an extended language (Clafer with behavior). Further, we have focused on the following five dimensions: support for configurable elements, including type systems, constraints, configuration, scalability, and formal semantics. Table 1 summarizes these results.

Together with [23], our overview provides an important resource for researchers as well as practitioners in the field of systems and software product line engineering on currently available textual variability modeling languages.

Given the large number of existing textual variability modeling languages, an obvious question is whether we need more languages. Our answer to this is as follows:

We do not just need further (textual) languages for existing concepts, leading to an even greater plethora of variability modeling languages, rather we need more innovative (textual) variability modeling approaches. As an indication for future directions, we discussed additional concerns in language design, including the choice of the first citizen concept, quantitative variability modeling, binding times, ecosystem support, exotic capabilities, and analyzability. In particular, we believe that future languages should have an extensible and modular language design with sub-languages catering for different needs, but holistically integrated into an over-arching concept.

ACKNOWLEDGEMENTS

This work is partially supported by the ITEA3 project REVaMP^2, funded by the BMBF (German Ministry of Research and Education) under grant 01IS16042H. Any opinions expressed herein are solely by the authors and not by the BMBF.

We thank the anonymous reviewers for their comments and suggestions that helped us improve the paper.

REFERENCES

[1] Andreas Abele, Yiannis Papadopoulos, David Servat, Martin Törngren, and Matthias Weber. 2010. The CVM Framework – A Prototype Tool for Compositional Variability Management. In *Proceedings of the 4th International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS'10) (ICB Research Report)*, David Benavides, Don S. Batory, and Paul Grünbacher (Eds.), Vol. 37. Universität Duisburg-Essen, 101–105.

[2] Mathieu Acher, Philippe Collet, Philippe Lahire, and Robert B. France. 2013. FAMILIAR: A domain-specific language for large scale management of feature models. *Science of Computer Programming* 78, 6 (2013), 657–681. https://doi.org/10.1016/j.scico.2012.12.004

[3] Ali Fouad Al-Azzawi. 2018. PyFML – A Textual Language For Feature Modeling. *International Journal of Software Engineering & Applications* 9, 1 (2018), 41–53. https://doi.org/10.5121/ijsea.2018.9104

[4] Mauricio Alférez, Mathieu Acher, José A. Galindo, Benoit Baudry, and David Benavides. 2019. Modeling variability in the video domain: language and experience report. *Software Quality Journal* 27, 1 (2019), 307–347. https://doi.org/10.1007/s11219-017-9400-8

[5] Timo Asikainen, Tomi Männistö, and Timo Soininen. 2006. A Unified Conceptual Foundation for Feature Modelling. In *Proceedings of the 10th International Software Product Line Conference (SPLC'06)*. IEEE, 31–40. https://doi.org/10.1109/SPLINE.2006.1691575

[6] Don S. Batory. 2005. Feature Models, Grammars, and Propositional Formulas. In *Proceedings of the 9th International Software Product Lines Conference (SPLC'05) (LNCS)*, Henk Obbink and Klaus Pohl (Eds.), Vol. 3714. Springer, 7–20. https://doi.org/10.1007/11554844_3

[7] Maurice H. ter Beek and Axel Legay. 2019. Quantitative Variability Modeling and Analysis. In *Proceedings of the 13th International Workshop on Variability Modelling of Software-intensive Systems (VaMoS'19)*. ACM, 13:1–13:2. https://doi.org/10.1145/3302333.3302349

[8] Maurice H. ter Beek and Axel Legay. 2019. Quantitative Variability Modeling and Analysis. *International Journal on Software Tools for Technology Transfer* (2019).

[9] Maurice H. ter Beek, Axel Legay, Alberto Lluch Lafuente, and Andrea Vandin. 2018. A framework for quantitative modeling and analysis of highly (re)configurable systems. *IEEE Transactions in Software Engineering* (2018). https://doi.org/10.1109/TSE.2018.2853726

[10] Maurice H. ter Beek, Axel Legay, Alberto Lluch Lafuente, and Andrea Vandin. 2018. QFLan: A Tool for the Quantitative Analysis of Highly Reconfigurable Systems. In *Proceedings of the 22nd International Symposium on Formal Methods (FM'18) (LNCS)*, Klaus Havelund, Jan Peleska, Bill Roscoe, and Erik de Vink (Eds.), Vol. 10951. Springer, 329–337. https://doi.org/10.1007/978-3-319-95582-7_19

[11] Thorsten Berger, Ştefan Stănciulescu, Ommund Øgård, Øystein Haugen, Bo Larsen, and Andrzej Wąsowski. 2014. To Connect or Not to Connect: Experiences from Modeling Topological Variability. In *Proceedings of the 18th International Software Product Line Conference (SPLC'14)*. ACM, 330–339. https://doi.org/10.1145/2648511.2648549

[12] Thorsten Berger, Steven She, Rafael Lotufo, Andrzej Wąsowski, and Krzysztof Czarnecki. 2013. A Study of Variability Models and Languages in the Systems Software Domain. *IEEE Transactions on Software Engineering* 39, 12 (2013), 1611–1640. https://doi.org/10.1109/TSE.2013.34

[13] Kacper Bąk, Krzysztof Czarnecki, and Andrzej Wąsowski. 2010. Feature and Meta-Models in Clafer: Mixed, Specialized, and Coupled. In *Proceedings of the 3rd International Conference on Software Language Engineering (SLE'10) (LNCS)*, Brian A. Malloy, Steffen Staab, and Mark van den Brand (Eds.), Vol. 6563. Springer, 102–122. https://doi.org/10.1007/978-3-642-19440-5_7

[14] Jan Bosch. 2009. From Software Product Lines to Software Ecosystems. In *Proceedings of the 13th International Software Product Line Conference (SPLC'09)*. Carnegie Mellon University, 111–119.

[15] Hendrik Brummermann, Markus Keunecke, and Klaus Schmid. 2012. Formalizing Distributed Evolution of Variability in Information System Ecosystems. In *Proceedings of the 6th International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS'12)*. ACM, 11–19. https://doi.org/10.1145/2110147.2110149

[16] Dave Clarke, Radu Muschevici, José Proença, Ina Schaefer, and Rudolf Schlatte. 2012. Variability Modelling in the ABS Language. In *Proceedings of the 9th International Symposium on Formal Methods for Components and Objects (FMCO'10) (LNCS)*, Bernhard Aichernig, Frank de Boer, and Marcello Bonsangue (Eds.), Vol. 6957. Springer, 204–224. https://doi.org/10.1007/978-3-642-25271-6_11

[17] Andreas Classen, Quentin Boucher, and Patrick Heymans. 2011. A text-based approach to feature modelling: Syntax and semantics of TVL. *Science of Computer Programming* 11, 12 (2011), 1130–1143. https://doi.org/10.1016/j.scico.2010.10.005

[18] Krzysztof Czarnecki, Paul Grünbacher, Rick Rabiser, Klaus Schmid, and Andrzej Wąsowski. 2012. Cool Features and Tough Decisions: A Comparison of Variability Modeling Approaches. In *Proceedings of the 6th International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS'12)*. ACM, 173–182. https://doi.org/10.1145/2110147.2110167

[19] Krzysztof Czarnecki, Simon Helsen, and Ulrich Eisenecker. 2005. Staged Configuration through Specialization and Multi-Level Configuration of Feature Models. *Software Process Improvement and Practice* 10, 2 (2005), 143–169. https://doi.org/10.1002/spip.225

[20] Eelco Dolstra, Gert Florijn, Merijn de Jonge, and Eelco Visser. 2003. Capturing Timeline Variability with Transparent Configuration Environments. In *ICSE Workshop on Software Variability Management (SVM'03)*, Peter Knauber and Jan Bosch (Eds.). IEEE. https://doi.org/10.1109/ICSE.2003.1201282

[21] Holger Eichelberger, Christian Kröher, and Klaus Schmid. 2013. An Analysis of Variability Modeling Concepts: Expressiveness vs. Analyzability. In *Proceedings of the 13th International Conference on Software Reuse (ICSR'13) (LNCS)*, John Favaro and Maurizio Morisio (Eds.), Vol. 7925. Springer, 32–48. https://doi.org/10.1007/978-3-642-38977-1_3

[22] Holger Eichelberger, Cui Qin, Roman Sizonenko, and Klaus Schmid. 2016. Using IVML to Model the Topology of Big Data Processing Pipelines. In *Proceedings of the 20th International Systems and Software Product Line Conference (SPLC'16)*. ACM, 204–208. https://doi.org/10.1145/2934466.2934476

[23] Holger Eichelberger and Klaus Schmid. 2015. Mapping the design-space of textual variability modeling languages: a refined analysis. *International Journal on Software Tools for Technology Transfer* 17, 5 (2015), 559–584. https://doi.org/10.1007/s10009-014-0362-x

[24] Sascha El-Sharkawy, Stephan Dederichs, and Klaus Schmid. 2012. From Feature Models to Decision Models and Back Again: An Analysis Based on Formal Transformations. In *Proceedings of the 16th International Software Product Line Conference (SPLC'12)*. ACM, 126–135. https://doi.org/10.1145/2362536.2362555

[25] Sascha El-Sharkawy, Adam Krafczyk, and Klaus Schmid. 2015. Analysing the Kconfig Semantics and Its Analysis Tools. In *Proceedings of the 14th International Conference on Generative Programming (GPCE'15)*. ACM, 45–54. https://doi.org/10.1145/2814204.2814222

[26] Martin Fowler. 2010. *Domain Specific Languages*. Addison-Wesley Professional.

[27] Paulius Juodisius, Atrisha Sarkar, Raghava Rao Mukkamala, Michał Antkiewicz, Krzysztof Czarnecki, and Andrzej Wąsowski. 2019. Clafer: Lightweight Modeling of Structure, Behaviour, and Variability. *The Art, Science, and Engineering of Programming* 3, 1 (2019), 2:1–2:62. https://doi.org/10.22152/programming-journal.org/2019/3/2

[28] Kyo C. Kang, Sholom G. Cohen, James A. Hess, William E. Novak, and A. Spencer Peterson. 1990. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. Technical Report CMU/SEI-90-TR-21. Carnegie Mellon University.

[29] KConfig Language [n.d.]. http://kernel.org/doc/Documentation/kbuild/kconfig-language.txt.

[30] Markus Keunecke, Hendrik Brummermann, and Klaus Schmid. 2013. The Feature Pack Approach: Systematically Managing Implementations in Software Ecosystems. In *Proceedings of the 8th International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS'14)*. ACM, 20:1–20:7. https://doi.org/10.1145/2556624.2556639

[31] Anna-Lena Lamprecht, Stefan Naujokat, and Ina Schaefer. 2013. Variability Management beyond Feature Models. *IEEE Computer* 46, 11 (2013), 48–54. https://doi.org/10.1109/MC.2013.299

[32] Marcilio Mendonça, Moises Branco, and Donald Cowan. 2009. S.P.L.O.T. – Software Product Lines Online Tools. In *Companion Proceedings of the 24th Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'09)*. ACM, 761–762. https://doi.org/10.1145/1639950.1640002

[33] Marcilio Mendonça, Andrzej Wąsowski, and Krzysztof Czarnecki. 2009. SAT-based Analysis of Feature Models is Easy. In *Proceedings of the 13th International Software Product Line Conference (SPLC'09)*. Carnegie Mellon University, 231–240.

[34] Rick Rabiser, Klaus Schmid, Martin Becker, Goetz Botterweck, Matthias Galster, Iris Groher, and Danny Weyns. 2018. A Study and Comparison of Industrial vs. Academic Software Product Line Research Published at SPLC. In *Proceedings of the 22nd International Systems and Software Product Line Conference (SPLC'18)*. ACM, 14–24. https://doi.org/10.1145/3233027.3233028

[35] Marko Rosenmüller, Norbert Siegmund, Thomas Thüm, and Gunter Saake. 2011. Multi-Dimensional Variability Modeling. In *Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems (VaMoS'11)*. ACM, 11–20. https://doi.org/10.1145/1944892.1944894

[36] Klaus Schmid. 2010. Variability Modeling for Distributed Development – A Comparison with established practice. In *Proceedings of the 14th International Conference on Software Product Line Engineering (SPLC'10) (LNCS)*, Jan Bosch and Jaejoon Lee (Eds.), Vol. 6287. Springer, 155–165. https://doi.org/10.1007/978-3-642-15579-6_11

[37] Klaus Schmid. 2013. Variability Support for Variability-Rich Software Ecosystems. In *Proceedings of the 4th International Workshop on Product LinE Approaches in Software Engineering (PLEASE'13)*. IEEE, 5–8. https://doi.org/10.1109/PLEASE.2013.6608654

[38] Klaus Schmid and Holger Eichelberger. 2008. Model-Based Implementation of Meta-Variability Constructs: A Case Study using Aspects. In *Proceedings of the 2nd International Workshop on Variability Modeling of Software-intensive Systems (VAMOS'08) (ICB Research Report)*, Patrick Heymans, Kyo C. Kang, Andreas Metzger, and Klaus Pohl (Eds.), Vol. 22. Universität Duisburg-Essen, 63–71.

[39] Klaus Schmid, Christian Kröher, and Sascha El-Sharkawy. 2018. Variability Modeling with the Integrated Variability Modeling Language (IVML) and EASy-producer. In *Proceedings of the 22nd International Systems and Software Product Line Conference (SPLC'18)*. ACM, 306–306. https://doi.org/10.1145/3233027.3233057

[40] Klaus Schmid, Rick Rabiser, and Paul Grünbacher. 2011. A Comparison of Decision Modeling Approaches in Product Lines. In *Proceedings of the 5th International Workshop on Variability Modeling of Software-intensive Systems (VaMoS'11)*. ACM, 119–126. https://doi.org/10.1145/1944892.1944907

[41] Pierre-Yves Schobbens, Patrick Heymans, and Jean-Christophe Trigaux. 2006. Feature Diagrams: A Survey and a Formal Semantics. In *Proceedings of the 14th International Requirements Engineering Conference (RE'06)*. IEEE, 139–148. https://doi.org/10.1109/RE.2006.23

[42] Arie van Deursen and Paul Klint. 2002. Domain-Specific Language Design Requires Feature Descriptions. *Journal of computing and information technology* 10, 1 (2002), 1–17. https://doi.org/10.2498/cit.2002.01.01

[43] Bart Veer and John Dallaway. [n.d.]. The eCos Component Writer's Guide. http://ecos.sourceware.org/docs-latest/cdl-guide/cdl-guide.html.

[44] Alexander von Rhein, Sven Apel, Christian Kästner, Thomas Thüm, and Ina Schaefer. 2013. The PLA Model: On the Combination of Product-Line Analyses. In *Proceedings of the 7th International Workshop on Variability Modelling of Software-intensive Systems (VaMoS'13)*. ACM, 14:1–14:8. https://doi.org/10.1145/2430502.2430522

[45] Jingang Zhou, Dazhe Zhao, Li Xu, and Jiren Liu. 2012. Do We Need Another Textual Language for Feature Modeling? A Preliminary Evaluation on the XML Based Approach. In *Software Engineering Research, Management and Applications 2012*, Roger Lee (Ed.). Studies in Computational Intelligence, Vol. 430. Springer, 97–111. https://doi.org/10.1007/978-3-642-30460-6_7