Comparing the Universal Variability Language with other Textual Variability Modeling Languages

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

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

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

Abstract

We compare the main features of the recently introduced Universal Variability Language (UVL) with other textual variability modeling languages from the software product line engineering domain. This comparison is structured according to the level of support that each language provides according to five dimensions: configurable elements, constraints, configuration, scalability, and formal semantics. This work extends our earlier work on comparing textual variability modeling languages that used a similar approach [7, 14].

CCS Concepts

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

Keywords

software product lines, variability modeling, textual specification languages, UVL

ACM Reference Format:
Maurice H. ter Beek, Klaus Schmid, and Holger Eichelberger. 2025. Comparing the Universal Variability Language with other Textual Variability Modeling Languages. In *29th ACM International Systems and Software Product Line Conference - Volume B (SPLC-B '25), September 01–05, 2025, A Coruña, Spain.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3748269.3749381

1 Introduction

Variability modeling research traditionally focused on graphical feature models [19] based on the concept of feature diagrams in the form of trees. This has led to a plethora of different notations, often varying only in minor technical details [27].

In practice, variability modeling has also been handled using textual notations. Prominent examples are KConfig [20] and CDL [29], which are textual variability description languages originating from the open-source domain. Unfortunately, such languages typically suffer from the fact that they are not fully (formally) defined and are therefore challenging to analyze [15].

Academia has also proposed a substantial number of textual variability modeling approaches [7], mostly following the notion of feature modeling with variations [27], yet sometimes following a decision modeling approach [26].

In this paper, we summarize and update the characterization of textual variability modeling languages presented in the First International Workshop on Languages for Modelling Variability (MODEVAR 2019) [7] with a comparison with the recently introduced Universal Variability Language (UVL) [8]. UVL, one of the primary outcomes of the MODEVAR initiative, was developed as a community effort during the previous eight workshop editions.[1]

Outline. In §2, we briefly introduce UVL, followed by an up-to-date overview of the state of the art of textual variability modeling languages in §3. In §4, we conclude and suggest some future work.

2 UVL: Universal Variability Language

The Universal Variability Language (UVL) [8] was created as a significant community effort, mostly driven in combination with the MODEVAR Workshop series.[1] It is meant to become a novel reference language for variability modeling research. Its design goals were: simplicity, information hiding, expressiveness, extensibility, and exchange (of models) [8].

An important characteristic of UVL is that it explicitly distinguishes multiple *levels* (cf. Table 1). The *core* language essentially corresponds to basic (Boolean) feature modeling with constraints, i.e., mandatory, optional, and alternative elements, attributes as well as (Boolean) constraints.

Language levels are subdivided into major levels (based on feature types) and minor levels. A minor level is always associated with a single major level and can optionally be included if the corresponding main level is included. The use of a specific language level is defined by the use of an `include` statement.

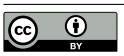
Major levels are strictly hierarchical, i.e., the use of the type level implies the inclusion of the arithmetic level. However, the use of minor levels is optional.

*All authors contributed equally to this paper.

[1] https://modevar.github.io/history/

Table 1: UVL Language characteristics (based on Fig. 3 in [8])

Major Level	Characteristics	Minor Level
Boolean	Boolean Features Group Keyword Boolean CTC Feature Attributes	Group Cardinality
Arithmetic	Numeric Constraints	Feature Cardinality Aggregate Function
Type	Type Features	String Constraints



SPLC-B '25, A Coruña, Spain

ACM ISBN 979-8-4007-2080-2/25/09
https://doi.org/10.1145/3748269.3749381

Despite the fact that UVL is rather new, it is already well supported by a number of *tools* from the research community for analysis as well as editing, such as FeatureIDE [21], flamapy [16], and variability.dev [17], according to [8].

3 Textual Variability Modeling Languages

In this section, we augment our earlier overview of textual variability modeling languages originally presented in [14], and updated in [7], with a first comparison with UVL, based on its presentation in [8].[2] We would like to emphasize that, in line with our earlier surveys, this paper focuses only on the expressiveness of the language. However, there always exists a tradeoff between expressiveness and analyzability. We refer the interested reader to our discussion in [12] and will not discuss analysis aspects further in this paper.

3.1 Literature Analysis Updated with UVL

The original literature analysis presented in [14, Sect. 4] considered the following textual variability modeling languages:

- *Feature Description Language (FDL)* [28] 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)* [22] 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)* [11] supports textual feature modeling, including capabilities for feature attributes, cardinalities and modularization.
- *μTVL* [10] is a variation of TVL, dropping some concepts, but also adding others like multiple trees in a single model.
- *CLAss, FEature, Reference approach (Clafer)* [9] combines meta-modeling of classes with feature modeling support.
- *VELVET* [24] is a language, inspired by TVL, but extends it in several directions and reimplements it from scratch.
- *Integrated Variability Modeling Language (IVML)* [25] follows the decision modeling paradigm with a strong focus on ease of learnability, expressiveness, and scalability.

The extended literature analysis presented in [7, Sect. 2] additionally considered the following textual variability modeling languages:

- *Clafer (extended with behavior)* [18] extends [9] 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 with specific constraints to ease reasoning on applications in the video domain, which was developed in an industrial project.

[2]While we are aware of further information on UVL, such as https://universal-variability-language.github.io/, we base our analysis mostly on [8], as it seems to be the most consistent and the most up-to-date resource.

In the meantime, the Universal Variability Language (UVL), as described in Section 2, was developed as a collective effort of the community [8]. Due to its importance, it is worthwhile to revisit the literature analysis and analyze UVL's position relative to other textual variability modeling languages.

We base our comparison of UVL with other textual variability modeling languages on the characteristics given in [7, Table 1], which are based on the following five dimensions:

- Configurable elements
- Constraint support
- Configuration support
- Scalability support
- Formal semantics

In Table 2, we show the classification of textual variability modeling languages from [7, Sect. 2], updated for UVL. Based on the dimensions mentioned above, we consider in detail (just as in [7, Table 1]) the following sub-dimensions:

Configurable elements and type systems: 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, such as simple dependencies, propositional logic, relational expressions, or arithmetic expressions.

Configuration support: default values, value assignment, and partial or complete configurations.

Scalability support: composition, i.e., the capability of integrating units of configurable elements into a single model.

Formal semantics: whether a detailed and formal definition of the semantics exists.

We refer the interested reader to [7, 14] for a detailed explanation of these criteria and further pointers to the literature that confirm the level of support offered by the surveyed textual variability modeling languages, except for UVL. Below, we will explain for each entry, the reasons for categorizing UVL the way we did. We will also use this as a basis for discussing UVL in relation to the other languages. The results are summarized in Table 2. As in [7], we simplified our notation to direct support (+), indirect support (±), unclear support (?) or no support (−).

An important challenge is that there is no single UVL language, but rather a multitude of sub-languages defined by the different language levels. Hence, we will discuss UVL in relation to its language levels. This is also annotated in Table 2, using a and t for arithmetic level and type level, respectively, and f and g for feature cardinality and group cardinality, respectively.

3.2 Configurable Elements

The basic configurable element in UVL is a *feature*, since the language follows the lead of FODA [19] rather closely. Thus, UVL supports mandatory, optional, and alternative features (and feature groups) directly. This is very much in line with more or less all other approaches. *Multiple* corresponds to group cardinalities, which in UVL is only available if the minor language level *group cardinalities* is included. The notion of feature cardinalities (i.e., replication of features) is not directly represented, in line with our earlier analysis.

Table 2: 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	+	+	+	−	+	+	−	+	−	−	+	+	−	+	+	+	+	+	±	+	−
UVL	+	+	$+^{g}$	+	+	$+^{g,f}$	−	$+^{t}$	−	−	−	+	−	$+^{a}$	$+^{a}$	−	−	−	−	+	−

Support: + = direct; ± = indirect; ? = unclear; − = no; *g*: group cardinality; *f*: feature cardinality; *t*: type level; *a*: arithmetic level

It is present in UVL only if major level *arithmetic* with minor level *feature cardinality* is included (cf. Table 1). Explicit support for extension mechanisms would imply that a variability model can later on be extended to cover variability that was not initially conceived. This is rather rare, as can be seen in Table 1. A limited notion of extension is available in UVL as it allows the combination of models, by mutual imports. This is similar to the approach taken by other languages.

Like most other languages, UVL also supports feature attributes. These include Boolean, Integer, Float, and String. In addition, attribute lists, which combine multiple attributes, are supported. However, UVL does not support explicit *typing* of attributes (cf. §3.3). This is (unfortunately) the case also for most other languages that support feature attributes. Further, at this point UVL does not support explicit *references* (cf. Section 3.6), which is surprising since about half of the languages we previously reviewed do support them.

3.3 Type Systems

When comparing UVL with other languages, one needs to clearly differentiate (typed) attributes from (typed) features. Although the first is already part of the base model of UVL, the latter is only added on the major level *Type*. In both cases, *predefined types* include Boolean, Integer, Float, and String. Attributes also include attribute lists as attributes, which one could see as separate types (similar to records). Based on the available information, it seems that there is no possibility for explicit typing of attributes and this is rather implicitly determined.

However, for features, there is the possibility of providing explicit typing at the major language level *Type*. However, this comes with a quirk: as stated in [8], features are selectable independent of the type. This means that they always have a Boolean component on top of the given type. Indeed, a Boolean feature is just selectable (identical to non-typed features on lower language levels), while other types include additional information. In the tradition of feature models, some other approaches do this in a similar way; however, as this basically overlaps with the notion of attributes with a slightly different syntax, it is unclear what the added benefit is.[3] This approach, while rather natural from a pure feature model perspective, also is not shared by all other languages. Some, like IVML [13, 14], do not implicitly add a Boolean type to typed information.

UVL also does not support *derived types*[4] or *user-defined types*, not even on major level *Type*. However, as shown in Table 2, support for such advanced typing is also rare in the other languages. Only FAMILIAR, Clafer, and IVML offer derived typing (essentially inheritance), while only Forfamel, TVL, Clafer, and IVML offer user-defined types.

3.4 Constraint Support

The various languages differ considerably in terms of their capabilities for *constraint expressions*. Different levels are shown in Table 2.

[3] It should also be pointed out that UVL defines conversion of types on feature level to attributes as part of its conversion strategies (cf. Table 2), if the Type level is not used.
[4] Derived types (as opposed to arbitrary user-defined types) refers to types based on other types, e.g., forms like inheritance.

While higher levels of expressiveness (both for configurable elements as well as constraints) significantly simplify or enable the modeling of complex situations, it typically also leads to a reduction of analyzability [12].

Like most other textual variability modeling languages, UVL does not support *simple constraint expressions* like 'requires', but directly goes for full *propositional logic*. It also provides higher levels of expressiveness like *relational* and *arithmetic constraints*. However, these are only available on major level *arithmetic*. On this level, UVL allows to represent also constraints over aggregates of attributes like the sum of all features need to have a size smaller than x. This is similar to capabilities of the other languages considered. In addition, UVL supports *string constraints*. This is only available as a separate minor level on major level typing. Hence, it can be regarded as the most advanced feature of UVL as it is on the highest level (cf. Table 1). While we did cover this in Table 2, this seems to be less supported by other languages. However, UVL support is also quite limited. It seems that only equality and length of strings are allowed.

Finally, like most other languages, UVL does not support any *first-order predicates* or other higher levels of expressiveness. Thus, the only languages that we know of that do explicitly support this, are Forfamel, VSL, Clafer, and IVML. Like all the other languages (except for IVML), UVL also does not support default constraints.

3.5 Configuration Support

UVL does not provide direct support for *configuration*, which is arguably the characteristic that distinguishes UVL most from the majority of other textual variability modeling languages.

Even one of the most elementary categories, *value assignment*, is not supported directly, whereas it is supported by all other languages except for FDL, Tree grammars, SXFM, and possibly μTVL. The difference is less marked for *default values*, which are supported by half of the other languages.

Except for FDL, Tree grammars, SXFM, TVL, and PyFML, all other languages support *configuration* as a first-class concept, allowing often arbitrary many configurations to be managed separately from the basic model description; moreover, except for Forfamel, these can be *partial configurations*. A recent paper on challenges for the industrial adoption of UVL specifically calls for the development of (tool) support for configurations, including generation, visualization, and verification [23]. Hence, this makes the issue that this is not an embedded language feature even more remarkable.

3.6 Scalability Support

Like most of the recent languages, UVL supports scalability through *composition*. Exceptions are PyFML as well as TVL and μTVL, which only allow *inclusion* and *conjunction*, respectively, of models.

UVL allows to *import* models and, moreover, to explicitly locate the imported feature tree (via an implicit reference) in the tree of the importing model.

Similar to UVL, VM supports the *import* of models and IVML supports the *scoped import* of models, which besides the provisioning of namespaces also provides scopes for the reasoning process, explicit versioning of imports, as well as a concept of interfaces.

Both Clafer and VELVET support scalability through *inheritence*, whereas FAMILIAR provides two explicit *composition* operators, namely the 'merge' and 'aggregate' operators for overlapping and disjoint models, respectively.

3.7 Formal Semantics

To establish the degree of formality of the *semantics* of the textual variability modeling languages, we consider whether they are given by examples, grammars, or a *formal* specification (or combinations thereof).

Fully *formal semantics* exist only for FDL, Forfamel (indirectly, by translation into a logical representation in Weight Constraint Rule Language), TVL, μTVL, and Clafer (extended with behavior) [7, 14].

Despite a dedicated section [8, Sect. 5: Syntax & semantics: Language specification], UVL does not have an unambiguous *formal semantics*. While the restrictions that different constraints in UVL impose on the configuration space (i.e., the set of valid, complete configurations modeled by a UVL model) are defined formally [8, Table 1: Constraint semantics], the semantics of basic feature modeling as well as additional capabilities like feature cardinality in UVL are only illustrated by examples that show how to resolve feature cardinality in a concrete UVL model [8, Listing 10: Cardinality Semantics Different Versions].

A further issue is that there does not seem to be a complete description of even the basic functionality of all the levels, let alone a definition which uniquely and formally defines the semantics for all allowed combinations of language levels. This is particularly striking as there is no definition of configuration within the language (cf. Section 3.5).

4 Conclusion

We presented an overview of the main characteristics of thirteen textual variability modeling languages in [7], as an extension of a systematic literature analysis reported in [14]. In this paper, we updated the characterization from [7] by considering also the recently introduced Universal Variability Language (UVL), focusing on the same five dimensions: whether the languages provide support for configurable elements, constraints, configuration, and scalability, and whether they are equipped with a formal semantics. Table 2 summarizes the outcome. On purpose, we refrained from any sort of rating, as we believe that each language's needs are made to fit the specific design goals of the specific language. Thus, a feature currently not present in UVL is not necessarily 'missing'.

Together with [7, 14], the overview and comparison serves as an important resource on all currently available textual variability modeling languages for both researchers and practitioners in the domain of systems and software product line engineering.

In [7], we aspired 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." UVL is such a language, sometimes with rather fine-grained minor levels (e.g., string constraints only contains one function and one predicate). So, UVL can clearly be seen as a major step forward, especially in terms of community unification. This already benefits the community significantly in terms of wealth of tool support.

The comparison presented in this paper also revealed significant potential for future work. For example, a precise definition of the syntax and semantics of UVL is still lacking, which is particular important given the potential level interactions. Also, the total range of capabilities is still rather limited, even on the highest level of the language, in comparison to other languages. The most striking omission in this area is the lack of explicit language-integrated configuration support. However, other advanced features like typing or references also strongly limit the range of applicability of UVL, developed to become the Universal Variability Language.

Acknowledgments

This work is partially supported by the German Federal Ministry of Research, Technology and Space (BMFTR) as DATIpilot innovation community ReGaP (grant 03DPC1511B) and by the Italian CNR project "Formal Methods in Software Engineering 2.0" (CUP B53C24000720005).

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, Vol. 37)*, David Benavides, Don S. Batory, and Paul Grünbacher (Eds.). Universität Duisburg-Essen, Germany, 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. doi: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. doi: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. doi: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, USA, 31–40. doi: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, Vol. 3714)*, Henk Obbink and Klaus Pohl (Eds.). Springer, Germany, 7–20. doi:10.1007/11554844_3

[7] Maurice H. ter Beek, Klaus Schmid, and Holger Eichelberger. 2019. Textual Variability Modeling Languages: An Overview and Considerations. In *Proceedings of the 23rd International Systems and Software Product Line Conference (SPLC'19)*, Vol. 2. ACM, USA, 82:1–82:7. doi:10.1145/3307630.3342398

[8] David Benavides, Chico Sundermann, Kevin Feichtinger, José A. Galindo, Rick Rabiser, and Thomas Thüm. 2025. UVL: Feature modelling with the Universal Variability Language. *Journal of Systems and Software* 225 (2025), 112326. doi:10.1016/J.JSS.2024.112326

[9] 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, Vol. 6563)*, Brian A. Malloy, Steffen Staab, and Mark van den Brand (Eds.). Springer, Germany, 102–122. doi:10.1007/978-3-642-19440-5_7

[10] 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, Vol. 6957)*, Bernhard Aichernig, Frank de Boer, and Marcello Bonsangue (Eds.). Springer, Germany, 204–224. doi:10.1007/978-3-642-25271-6_11

[11] 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. doi:10.1016/j.scico.2010.10.005

[12] 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, Vol. 7925)*. Springer, Germany, 32–48. doi:10.1007/978-3-642-38977-1_3

[13] 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, USA, 204–208. doi:10.1145/2934466.2934476

[14] 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. doi:10.1007/s10009-014-0362-x

[15] 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, USA, 45–54. doi:10.1145/2814204.2814222

[16] José A. Galindo, José Miguel Horcas, Alexander Felfernig, David Fernández-Amorós, and David Benavides. 2023. FLAMA: A collaborative effort to build a new framework for the automated analysis of feature models. In *Proceedings of the 27th International Systems and Software Product Line Conference (SPLC'23)*, Vol. 2. ACM, USA, 16–19. doi:10.1145/3579028.3609008

[17] Tobias Heß, Lukas Ostheimer, Tobias Betz, Simon Karrer, Tim Jannik Schmidt, Pierre Coquet, Sean Semmler, and Thomas Thüm. 2024. variability.dev: Towards an Online Toolbox for Feature Modeling. In *Proceedings of the 6th International Workshop on Languages for Modelling Variability (MODEVAR'24)*. arXiv, USA. doi:10.48550/arXiv.2506.09845

[18] 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. doi:10.22152/programming-journal.org/2019/3/2

[19] 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, USA.

[20] KConfig Language. 2018. http://kernel.org/doc/Documentation/kbuild/kconfig-language.txt.

[21] Jens Meinicke, Thomas Thüm, Reimar Schröter, Fabian Benduhn, Thomas Leich, and Gunter Saake. 2017. *Mastering Software Variability with FeatureIDE*. Springer, Switzerland. doi:10.1007/978-3-319-61443-4

[22] 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, USA, 761–762. doi:10.1145/1639950.1640002

[23] Rick Rabiser. 2024. Industry Adoption of UVL: What We Will Need. In *Proceedings of the 28th ACM International Systems and Software Product Line Conference (SPLC'24)*, Vol. 2. ACM, USA, 46–49. doi:10.1145/3646548.3676597

[24] 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, USA, 11–20. doi:10.1145/1944892.1944894

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

[26] 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, USA, 119–126. doi:10.1145/1944892.1944907

[27] 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, USA, 139–148. doi:10.1109/RE.2006.23

[28] 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. doi:10.2498/cit.2002.01.01

[29] Bart Veer and John Dallaway. 2001. The eCos Component Writer's Guide. http://ecos.sourceware.org/docs-latest/cdl-guide/cdl-guide.html.