Yet Another Textual Variability Language? A Community Effort Towards a Unified Language

Chico Sundermann
University of Ulm
Ulm, Germany
chico.sundermann@uni-ulm.de

Kevin Feichtinger
LIT CPS Lab
Johannes Kepler University Linz
Linz, Austria
kevin.feichtinger@jku.at

Dominik Engelhardt
TU Braunschweig
Braunschweig, Germany
d.engelhardt@tu-bs.de

Rick Rabiser
CDL VaSiCS, LIT CPS Lab
Johannes Kepler University Linz
Linz, Austria
rick.rabiser@jku.at

Thomas Thüm
University of Ulm
Ulm, Germany
thomas.thuem@uni-ulm.de

ABSTRACT

Variability models are commonly used to model commonalities and variability in a product line. There is a large variety of textual formats to represent and store variability models. This variety causes overhead to researchers and practitioners as they frequently need to translate models. The MODEVAR initiative consists of dozens of researchers and aims to find a unified language for variability modeling. In this work, we describe the cooperative development of a textual variability language. We evaluate preferences of the community regarding properties of existing formats and applications for an initial design of a unified variability language. Then, we examine the acceptance of the community for our proposal. The results indicate that our proposal is a promising start towards a unified variability language instead of *yet another language*. We envision that the community applies our language proposal in teaching, research prototypes, and industrial applications to further evolve the design and then ultimately reach a unified language.

CCS CONCEPTS

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

KEYWORDS

variability modeling, variability language, unified language, exchange format, software product lines

ACM Reference Format:
Chico Sundermann, Kevin Feichtinger, Dominik Engelhardt, Rick Rabiser, and Thomas Thüm. 2021. Yet Another Textual Variability Language? A Community Effort Towards a Unified Language. In *25th ACM International Systems and Software Product Line Conference - Volume A (SPLC '21), September 6–11, 2021, Leicester, United Kingdom.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3461001.3471145

SPLC '21, September 6–11, 2021, Leicester, United Kingdom

ACM ISBN 978-1-4503-8469-8/21/09…$15.00
https://doi.org/10.1145/3461001.3471145

1 INTRODUCTION

Product lines are widely used to manage families of similar products [14]. Variability models are common to specify constraints of a product line. However, there are different types of variability models, such as feature models and decisions models. The literature also considers a large variety of textual representations to describe variability models [1, 3, 4, 6, 9, 22, 23, 27, 41, 44, 49, 55, 59, 60, 76]. Furthermore, existing tools typically depend on specific formats [3, 15, 44, 49, 78]. Working with a multitude of formats induces large additional effort for practitioners and researchers. Using a variability model that is stored in a different format requires implementing a parser first. For example, if a specific capability (e.g., to analyze a model) is only available in a particular tool, variability models need to be translated to a format that is supported by the specific tool. If the community would widely adopt a single format, the described efforts could be greatly reduced. Even though there have been standardization efforts (e.g., ISO-43117 [1] and the Common Variability Language (CVL) [37]), those either failed or have not been picked up by the community due to legal reasons and insufficient involvement of the community. Nevertheless, universal formats can have (some) success as demonstrated by XML [20], JSON [54], UML [33], SMT [7], and DIMACS [56].

Our insights indicate that the community desires a unified format. The MODEVAR initiative, which was started by David Benavides through a letter to representatives of the community in September 2018, aims on creating a unified variability modeling language [13]. There have been already three MODEVAR workshops in the last two years (i.e., one in 2019 and two in 2020) with dozens of participants and several publications on modeling variability [10, 11, 17, 31, 35, 42, 51, 58, 75, 77]. Originated in the MODEVAR community effort, we have build a first language proposal for a unified language.

In contrast to other works presenting variability languages, we focus on evaluating the preferences of the community and finding a widely accepted base of concepts for a textual notation. We

[1] https://www.iso.org/obp/ui/#iso:std:43117:en

performed two questionnaire-based surveys with students and variability modeling researchers including leading domain experts. We use our results to introduce the first proposal of a variability language that was developed in tight cooperation with the community. Our resulting first proposal, the Universal Variability Language (UVL), is primarily intended to be a common textual exchange format. However, for the language design, we considered a variety of additional requirements (e.g., the format should also be human editable and suitable for teaching). The insights of the questionnaires indicate that our proposal is a promising starting point and could be widely accepted by the community. Nevertheless, we are aware that there is no guarantee of UVL being widely adopted. To tackle issues of previous proposals, we develop tooling and language design openly (i.e., provide open-source repositories with licenses allowing wide usage and distribution, see footnotes 2 and 3) and encourage the involvement of the community. We do not consider UVL's design as final but as a base that we plan to advance with the community and in particular with the MODEVAR initiative.

To simplify the usage of UVL, we provide a default library that supports the parsing and printing of UVL. [2] The library can be used either as standalone tool or embedded into other tools. Currently, we already integrated UVL in the popular feature modeling tool FeatureIDE [44] and the transformation approach TRAVART [32], which now uses UVL as pivot language, enabling automatic transformations from/to several existing variability modeling notations.

In addition to the language design and tool support, we provide a dataset of real-world variability models in the UVL format.[3] For each variability model, we provide additional information as suggested by Galindo and Benavides [35], such as source of the model, multiple versions (if available), and model metrics.

In summary, we make the following **contributions** to provide a base for a universal textual format for variability models:

(1) Concepts to consider when designing a variability language
(2) A questionnaire-based survey to evaluate the community's preferences on the gathered concepts
(3) A language design based on the preferences (UVL)
(4) A questionnaire-based survey to evaluate the community's acceptance of UVL
(5) A database of real-world variability models in UVL

This work is structured as follows. In Section 2, we provide a running example and give a short introduction to feature models, as the most prominent type of variability models. In Section 3, we gather structural properties and application domains for variability languages. Then, we evaluate the preferences of the community regarding the gathered properties. In Section 4, we propose the Universal Variability Language (UVL) and explain our design decisions, which are based on community feedback. We provide two different variants for comparison to acquire more precise results on the acceptance of the community. In Section 5, we evaluate UVL on three aspects: (1) how well it satisfies the presented application scenarios, (2) the acceptance of the community and their favored variant, and (3) the scalability when applied to industrial variability models. In Section 6, we discuss related work. In Section 7, we conclude and debate future work.

[2] http://doi.org/10.5281/zenodo.5031677
[3] http://doi.org/10.5281/zenodo.5031829

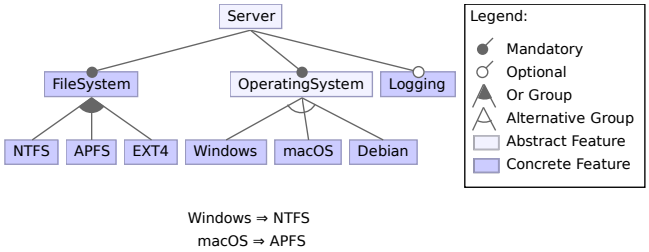

Figure 1: Example Feature Model [30]

2 BACKGROUND AND RUNNING EXAMPLE

A plethora of variability modeling approaches representing the common and variable characteristics of a set of (software) systems have been proposed in the last 30 years [57]. Numerous case studies show variability models are widely applied in academia and industry [18, 47]. Among all these approaches, feature models are probably the most common approach to specify the constraints of a configurable system [24]. For better readability, we describe our running example with the help of a feature diagram.

A feature diagram is a visual representation of a feature model. Figure 1 shows a feature diagram representing a simplified server. Each `Server` requires a `FileSystem` and an `OperatingSystem` (denoted by a MANDATORY flag). In addition, a server may have `Logging` (denoted by OPTIONAL flag). The server requires at least one child feature of `FileSystem` (OR group) and exactly one child feature of `OperatingSystem` (ALTERNATIVE group). Furthermore, the cross-tree constraints below the feature tree indicate that `Windows` and `macOS` require the file systems `NTFS` and `APFS`, respectively. As feature diagrams and generally graphical representations are out of the scope of this work, we refer the reader to other works for more details [9, 14, 19, 26, 61, 62].

While the basic graphical notation (i.e., feature diagrams) is widely adopted [24], there is a large variety of textual notations. These textual formats specify the properties of a variability models using many different structural concepts. In the following, we use Figure 1 as running example to explain the considered concepts.

3 CHARACTERISTICS OF VARIABILITY LANGUAGES

Our main goal is to find a community consensus when designing the variability language. Thus, we designed UVL with recurring feedback from the community. In this section, we analyze (1) application scenarios for variability languages proposed by a multitude of variability modeling experts [17] in Section 3.1 and (2) 14 existing textual representations [1, 3, 4, 6, 9, 22, 23, 41, 44, 49, 59, 60, 76] for feature models in Section 3.2 to identify the critical and accepted structural elements of the new language. We expect that reusing existing concepts yields two benefits, namely easier access for the community [43] and widely accepted notations. In Section 3.3, we present additional language features we identified to set a scope for UVL. After collecting the concepts, we presented them to the community and evaluated their preferences using a questionnaire which we discuss in Section 3.4.

3.1 Application Scenarios

Berger and Collet [17] summarize 15 usage scenarios for textual feature-model notations which were collected with a group of variability modeling experts. The collected scenarios were then evaluated regarding their usefulness by 15 participants in an online questionnaire. We use their insights to design UVL for applications deemed as useful and important by the community. In the following, we explain each scenario that we consider relevant for designing the language in contrast to scenarios either not dependent on the language or deemed as not useful by the community. We sort the relevant scenarios by their usefulness, rated by the community [17], in a descending order (i.e., most useful scenarios first).

Exchange. Feature models should be transferable from one tool to another. The language should fulfill properties to simplify importing and exporting, such as a serializable syntax, sufficient documentation, and allow storing tool-specific data. Furthermore, a parser library that supports importing and exporting should be provided.

Mapping to Implementation. Typically, some features in a feature model correspond to implementation artifacts, such as source code. Features that correspond to such an artifact should be distinguishable from features that are just used for structuring. The mapping may be realized by assigning features to specific artifacts or just by a binary keyword that indicates a purely structural feature (e.g., abstract features in FeatureIDE [48] or compound in GUIDSL [9]).

Teaching and Learning. It should be as simple as possible to understand and learn the syntax of the language. For example, the format should be explainable on a few slides [17]. The language should use concepts that are prevalent in computer science education for easier access. Furthermore, realistic examples and toy examples should be provided.

Storage. The tool should allow (1) sparse storing and (2) efficient loading for feature models. For (1), the language should have a succinct syntax. For (2), the language should be specified in a format that allows simple implementation of efficient parsers, such as context-free grammars.

Decomposition and Composition. Industrial variability models can come with several thousands of features [40, 46, 47, 72]. The sheer size of the model then can lead to major difficulties when working with graphical or textual representations. In addition, edits to a variability model are typically limited to a subset of features. One considered solution is decomposing the variability model into multiple sub-models which typically simplifies editing and reading. A variability language should thus support decomposing variability models and also composing the resulting sub-models.

Overall, we considered five out of 14 application scenarios for the design of our language. First, we filtered scenarios that are not relevant for designing the language as they are dependent on additional tooling instead of the language itself. For that reason, we omitted seven scenarios, namely *benchmarking*, *domain modeling*, *analyses*, *model generation*, *testing*, and *reverse engineering*. Second, we filtered two scenarios that are not regarded to be useful as part of the language by the community, namely *configuration*, *writing, reading, and editing*, and *model weaving*. For details on the omitted scenarios, we refer to the original work [17].

```
Server {
  FS {
    or { NTFS APFS }
  }
  OS {
    ...
  }
}
```

(a) Nesting Brackets

```
Server
  FS
    or
      NTFS
      APFS
  OS
    alternative
      ...
```

(b) Nesting Indentation

```
Server: FS OS Log;
FS: or NTFS APFS EXT4;
OS: alt Windows Debian;
```

(c) Reference

Figure 2: Specification of Hierarchy

```
FS: or NTFS APFS EXT4;
OS: alternative Windows Debian;
```

(a) Long Keywords

```
FS: or NTFS APFS EXT4;
OS: alt Windows Debian;
```

(b) Abbreviations

```
FS: (NTFS | APFS | EXT4)+;
OS: Windows | Debian;
```

(c) Symbols

Figure 3: Examples for Keywords

3.2 Structural Characteristics

When analyzing existing notations [1, 3, 4, 6, 9, 22, 23, 41, 44, 49, 59, 60, 76], we identified six structural categories to characterize the different languages. We use those to identify popular concepts for textual notations. Table 1 gives an overview on the analyzed languages regarding the extracted categories.

Hierarchy. Each of the considered textual formats uses either references [3, 9, 76] or nesting [1, 4, 6, 22, 23, 41, 44, 49, 59, 60] to describe the hierarchy of the feature model. For references, each feature is referred to by an identifier (i.e., the feature name) which is used to declare parent-child relationships (cf. Figure 2c). For nesting, the parent-child relationship is declared by defining the children within a block of their parent (cf. Figure 2b).

Blocks. If nesting is used to define the hierarchy for a feature model, a notation is required to declare a block. All but one language we analyzed that uses a nesting hierarchy uses either indendation (cf. Figure 2b) [41, 49], or different types of brackets (cf. Figure 2a) [1, 4, 6, 22, 23, 59, 60], to define blocks. The only exception is FeatureIDE which uses XML tags to define blocks [44]. While SXFM also uses XML, the format uses indentation to specify blocks and only uses tags to separate the tree-hierarchy from the cross-tree constraints [49].

Keywords. The usage of keywords in the analyzed language can be categorized into three groups: First, the keyword can be fully written (e.g., "alternative", cf. Figure 3a) [4, 22, 23, 59, 60, 76]. Second, the keyword can be abbreviated (e.g., "alt" for alternative, cf. Figure 3b) [1, 6, 41, 44]. Third, symbols can be used (e.g., "|" for an OR-group, Figure 3c) [3, 9, 49].

Table 1: Summary of Structural Characteristics of the Analyzed Languages

Language	Hierarchy	Blocks	Keywords	Line Endings	Constraint Location	Location of Groups
FDL	reference	none	full	new line	separate	parent
GUIDSL	reference	none	symbols	semicolon	separate	parent[1]
FAMILIAR	reference	none	symbols	semicolon	separate	parent
PyFML	nesting	[]	full	semicolon	separate	parent
Clafer	nesting	indentation	minimal	new line	in-line	parent
SXFM	nesting	indentation	symbols	new line	separate	between
VSL	nesting	()	minimal	semicolon	separate	between
TVL	nesting	{}	full	semicolon	in-line	between
μTVL	nesting	{}	full	semicolon	in-line	between
VELVET	nesting	{}	full	semicolon	in-line	between
VM	nesting	{}	minimal	new line	separate	between
IVML	nesting	{}	full	semicolon	in-line	n/a
FeatureIDE	nesting	XML	full	XML	separate	between

[1] for *or* groups also the parent's parent

```
Server
  or FS
    NTFS
    APFS
  alternative OS
    Windows
    Debian
```

(a) At Parent

```
Server
  FS
    or
      NTFS
      APFS
  OS
    alternative
      ...
```

(b) Inbetween

Figure 4: Examples for Groups Location

```
Server
  FS
    ...
  OS
    alternative
      Windows
        Windows => NTFS
      Debian
        Debian => APFS
  Logging ...
```

(a) Inline Constraints

```
Server
  FS
    or
      ...
  OS
    alternative
      ...

constraints
  Windows => NTFS
  macOS => APFS
```

(b) Separate Constraints

Figure 5: Examples for Constraints Location

Line Endings. For each analyzed language, the end of a statement is either specified by a new line (cf. Figure 2b) [6, 41, 49, 76], or a semicolon (cf. Figure 2c) [1, 3, 4, 9, 22, 23, 59, 60]. The only exception is FeatureIDE which uses XML tags for the end of a statement [44].

Location of Groups. The group describing the parent-child relationship is stored either at the parent (cf. Figure 4a) [3, 4, 9, 41, 44, 76] or within the list of children (cf. Figure 4b) [1, 6, 22, 23, 49, 59].

Cross-Tree Constraint Location. There are two variants for storing cross-tree constraints in the analyzed languages. First, the constraints can be stored in-line next to declarations of features referenced by the constraint (cf. Figure 5a) [22, 23, 41, 59, 60]. Second, the constraints can be stored separately, typically below the block describing the feature tree (cf. Figure 5b) [1, 3, 6, 9, 49, 76].

3.3 Language Scope

In addition to the application scenarios and structural characteristics, we identified several language features that may be part of a variability language's scope. Those features either appear in at least one of the analyzed variability languages or are mentioned in the application scenarios [17].

Default Selections. When configuring product variants, some features may be selected or deselected by default. The default selection of features could be realized by attaching an indicator to the feature.

Abstract Features. An abstract feature does not correspond to any implementation artifact but is used only for structuring. An abstract feature could be marked with a flag [9, 44].

Save Entire Configurations. In some cases, it may be interesting to store entire default configurations (e.g., an Ubuntu configuration for the Linux kernel). Those could be stored persistently in the variability model.

Attributes for Features. Non-functional attributes could be attached to the features. For example, a numerical attribute may indicate the price of its feature.

References to Other Variability Models. A reference to other variability models could be used as a simple mechanism to enable composition. Then, the referenced variability model could be integrated as subtree of the composed variability model.

Feature Model Interfaces. When composing variability models, a feature model interface can be used to only expose a subset of the features and constraints of the referenced variability model [63].

Extension Mechanism for Arbitrary Data. Tool-specific data (e.g., preferences or layouting information) may be required to be stored. A generic extension mechanism for a variability language may simplify storing such data.

Table 2: Community Feedback for Language Scenarios

Category	#Votes
Keywords	**Complete** (5)
Line Breaks	Semicolon (5), **New line** (5)
Structuring	**Indendation** (6), Curly Braces (4)
Hierarchy	**Reference** (5), **Nesting** (4)
Group Location	**In-between** (7)
Group Mechanism	**Cardinality** (7), **Groups** (6)
Constraints	**Propositional Logic** (9)
Tool Support	**Default Library** (7), **EBNF** (6)
Reuse Format (e.g., XML)	Yes (4), **No** (4)

3.4 Preferences of the Community

We performed a questionnaire with 20 participants at the MODEVAR workshop at VaMoS'20.[4] The participants are all involved in variability-modeling research. Furthermore, there are multiple leading experts on variability modeling among the participants.[5] In the first part of the questionnaire, the participants answered questions regarding their preferences on the collected structural characteristics described in Section 3.2. In the second part, the participants decided which language features (cf. Section 3.3) should be included in a language according to the community. While answering the questionnaire, the participants worked in pairs to discuss their positions and provide more well-thought-out answers. Thus, there are overall 10 submissions from the 20 participants.

Table 2 shows the results for the first part of the questionnaire. For each category, the table displays all answers that received at least one third of the votes. The participants clearly favor complete keywords, location of groups in-between, and propositional constraints. Specifying the parent-child relationship with cardinality and groups (e.g., alternative) received seven and six votes, respectively. The textual comments indicate that most participants prefer that a language supports both concepts. Furthermore, the tooling should provide a default library with a parser and printer, and an EBNF specification. The participants slightly favor indentation (six votes) over curly braces to declare blocks (four votes). For line breaks, the participants are split between a semicolon and new line. However, in the textual comments semicolons are mostly regarded as redundant and a concept that is only familiar to programmers. For specifying hierarchy, references and nesting received five and four votes, respectively. The decision is split on whether existing formats (e.g., XML, JSON, or YAML) should be used. In the textual comments, the participants stated that reusing serialized formats simplifies parsing but hinders readability and, thus, should be mainly used for sole exchange formats.

Figure 6 shows the results for the second part of the questionnaire. For abstract features, feature attributes, references to other feature models, and an extension mechanism, at least half of the participants would prefer for them to be part of UVL and neither participant is strictly against it (i.e., voted absolutely not). While

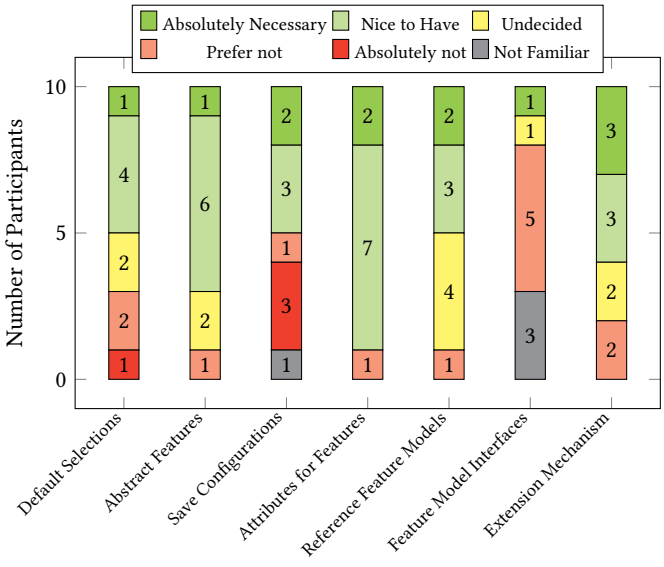

Figure 6: Voting Results on Inclusion of Language Features

no participant voted *absolutely not* for feature model interfaces, only two voted for absolutely necessary or nice to have. Default selections and saving entire configurations are disputed as three and four participants prefer to exclude the concept, respectively.

In summary, we design the structural characteristics of UVL based on the following insights: Full-length keywords are preferred over abbreviations and symbols. For defining blocks, indendation is more popular. As mechanism to describe groups, both cardinalities and group descriptors, such as OR, should be available. The participants are split on whether to use references or nesting to describe the hierarchy of the feature model. While semicolons and line-breaks received equally many votes, the participants argue that semicolons are redundant and only make sense for programmers. We assume that often it won't be programmers who create or maintain variability models and, thus, select line-breaks to specify the end of a statement. Furthermore, the language should support abstract features, feature attributes, default selections for configuring, simple references to enable decomposition, and an extension mechanism to attach tool-specific data to elements. Propositional logic is regarded as sufficient for cross-tree constraints. The tooling support should contain a library that supports reading and printing, and a specification as EBNF. Based on the textual feedback, we decided not to reuse a serialization format as UVL should be human editable in contrast to a sole exchange format.

4 UNIVERSAL VARIABILITY LANGUAGE

We use the insights gathered in Section 3 for a first proposal of the Universal Variability Language (UVL). As the participants were undecided whether to use nesting or references as hierarchy mechanism, we provide two alternative variants for the community to evaluate. In Section 4.1, we present the variant using nesting. In Section 4.2, we discuss the variant using references by showing the differences to the nesting variant. During development, we considered both variants as equally suitable candidates for UVL. However,

[4] https://modevar.github.io/vamos-2020/

[5] Leaving the name was optional. Two teams of participants decided to not share their names. Participants who shared their name with highly influential works on variability modeling: David Benavides [14, 16], Maurice ter Beek [73, 74], Mathieu Acher [2, 3], Gilles Perrouin [52, 53], José A. Galindo [65]

Listing 1: Running Example in UVL

```
features
  Server {abstract}
    mandatory
      FileSystem
        or // with cardinality: [1..*]
          NTFS
          APFS
          EXT4
      OperatingSystem {abstract}
        alternative
          Windows
          macOS
          Debian
    optional
      Logging {
      default,
      log_level "warn" // Feature Attribute
    }

constraints
  Windows => NTFS
  macOS => APFS
```

we later performed a questionnaire-based survey (cf. Section 5) indicating that the variant using nesting is clearly favored. Thus, we consider the nesting variant as the UVL.

4.1 Variant 1: Nesting

We provide a context-free grammar in EBNF notation, further explanations and more examples for UVL at the parser library repository[6]. In the following, we describe our decisions on the language design based on the categories gathered in Section 3.2. Listing 1 shows our running example specified in UVL (i.e., the nesting variant).

Hierarchy. The hierarchy of the feature model is specified using nesting. Each group describing the parent-children relationship (e.g., ALTERNATIVE or MANDATORY) is specified in an indented block.

Complete Keywords. We use complete keywords for our language, in contrast to abbreviations or symbols. We support the following keywords to specify parent-child relationships: `alternative`, `or`, `optional`, and `mandatory`.

Cardinalities and Groups. In addition to groups (such as ALTERNATIVE), we support cardinalities to describe the parent-child relationship. Cardinalities can be described as follows: `[a..b]` or `[b]` where `a` is an integer and `b` is an integer or a wildcard (* in UVL).

Separate Constraints. We store cross-tree constraints separately from the tree structure. The block specifying constraints is marked with a `constraints` keyword as header.

Propositional Constraints. For the current version of UVL, cross-tree constraints are limited to propositional logic, contrary to first-order logic. The following logical operators are supported: ¬ (`!` in UVL), ∧ (`&`), ∨ (`|`), ⇒ (`=>`), and ⇔ (`<=>`).

Scope. UVL supports abstract features, feature attributes, and decomposition & composition of feature models. Furthermore, feature attributes can be used as an extension mechanism to store tool-specific data. The `abstract` flag and feature attributes are specified in curly brackets behind the features, possibly over multiple lines (cf. `Logging` in Listing 1).

[6] http://doi.org/10.5281/zenodo.5031677

Listing 2: Decomposition Example FileSystem

```
features
  FileSystem
    or
      NTFS
      APFS
      EXT4
```

Listing 3: Composition Example in UVL

```
imports
  FileSystem as fs // import from namespace
  OperatingSystem as os
features
  Server {abstract}
    mandatory
      fs.FileSystem
      os.OperatingSystem
    optional
      Logging ...

constraints
  os.Windows => fs.NTFS
  ...
```

Decomposition & Composition. UVL comes with a composition mechanism which is similar to Java packages and imports. Listing 2 shows the decomposed feature model of the file system. Listing 3 shows an example of composing the Server feature model. Here, the required submodels are imported with the `imports`-flag. With the `as`-keyword a namespace alias can be provided (e.g., `fs` for the file system). Then, a feature appearing in the submodel can be referenced in the hierarchy or in constraints with that alias (e.g., `fs.FileSystem`).

4.2 Variant 2: References

The major difference between the two variants is the specification of hierarchy. The second variant uses references while the first variant uses nesting. Listing 4 shows our running example specified with the references variant for the hierarchy, to highlight the differences. Every line defining a feature specifies (1) its children, (2) the parent-child relationship (e.g., OR), (3) feature attributes, and (4) an indicator whether the feature is abstract. Mandatory features are marked with an exclamation mark. Attributes and the abstract-flag are separated with a comma from the children.

Listing 4: UVL Variant with References

```
features
Server: FileSystem! OperatingSystem! Logging, abstract
FileSystem: or NTFS APFS EXT4
OperatingSystem: alternative Windows macOS Debian, abstract
Logging
  default
  log_level "warn"

constraints
Windows => NTFS
macOS => APFS
```

4.3 Tooling

We provide a parser library for UVL[7] that supports reading and writing the format with respect to (1) feature attributes, (2) abstract features, and (3) decomposition of feature models. The tool can be either used standalone or embedded into existing tools. We integrated our UVL parser into FeatureIDE [48] (available for v3.7.1) and TRAVART [32]. With the integration in FeatureIDE, UVL is available and usable in a popular and maintained tool that is commonly used in research and industry [5, 32, 34, 38, 45, 70]. FeatureIDE supports reading and storing UVL feature models including composed models. The graphical user interface can also be used to display and edit UVL models. For more details and a discussion on the integration into FeatureIDE, we refer to our recent work [71]. TRAVART can be used to automatically transform variability models (i.e., feature and decision [27] models) with a variety of different formats to UVL and vice versa [32]. Thus, we expect that the integration of UVL in TRAVART vastly simplifies the usage of UVL.

We decided to specify UVL using a *context-free grammar* (CFG) with EBNF notation, which allows the usage of widely adopted tools (e.g., ANTLR [50]) to generate a parser. Furthermore, the language can be easily adapted/extended and we assume that many users are already familiar with CFGs. Our tool uses *instaparse*[8] to specify the language with a context-free grammar.

5 EVALUATION

The goal of the evaluation is to examine the suitability of UVL as a widely accepted format. The evaluation is structured as follows. First, we inspect how well UVL satisfies the application scenarios proposed by the community [17] (cf. Section 3.1). Second, we examine the acceptance of the community for our proposal of UVL. Here, we evaluate which of the two variants is seen favorable and, then, give more insights on the detailed feedback of the community. Third, we perform a technical evaluation that inspects the scalability of UVL when applied to real-world variability models with regards to size and export time. We also discuss potential threats to validity. Overall, we investigate the following research questions:

RQ1 Does UVL satisfy the presented application scenarios?
RQ2a Which variant of UVL is preferred by the community?
RQ2b How well is UVL accepted by the community?
RQ3 Does UVL scale to real-world variability models?

5.1 Evaluating Against Requirements

Regarding **RQ1**, we discuss in this subsection how UVL satisfies the application scenarios presented in Section 3.1.

Exchange. UVL satisfies the requirements proposed by the community for exchange. We provide documentation in the UVL repository, a context-free grammar that specifies all possible elements, and give a detailed explanation in this work. Feature attributes allow a variety of extensions (e.g., for tool-specific data) due to their flexible nature. In addition, we provide an open-source parser library to enable reading and writing of UVL.

Mapping to Implementation. UVL can integrate the mapping to implementation twofold. First, features that do not correspond to any artifacts can be marked with an abstract flag. Second, additional arbitrary information about the artifacts or the mapping can be stored using feature attributes.

Teaching and Learning. UVL is based on many widely used concepts from computer science, such as the namespaces which are inspired by the Java import system. The language formatting is similar to Python syntax which should be familiar to many users. Furthermore, the structure is similar to graphical feature-model notations with a tree structure and separate cross-tree constraints.

Storage. UVL uses a sparse format that comes without overhead typically induced by more generic formats, such as XML which is currently used by FeatureIDE [44]. Furthermore, the results of our empirical evaluation (cf. Section 5.3) indicate that UVL is one of the formats that require the least space to store a feature model.

Decomposition and Composition. UVL comes with a mechanism to compose feature models. Submodels can be specified in separate files and then integrated with a syntax similar to the Java import system. We also integrated support for the composition mechanism in FeatureIDE, so it can be used with a graphical user interface.

Discussion. Regarding **RQ1**, our analysis of the requirements strongly indicates that UVL satisfies each proposed application scenario, which is relevant for the language design and deemed as useful by the community.

5.2 Acceptance of the Community

In this section, we present our insights on the acceptance of the community for UVL to answer **RQ2a** and **RQ2b**.

Experiment Design. We performed an online questionnaire-based survey to evaluate the acceptance of the community for both language variants of UVL (cf. Section 4). In this paper, we introduce the nesting variant as UVL. However, before and during the survey, we considered both variants as equal candidates for UVL. The decision to select the nesting variant as UVL is based on the results of the questionnaire. In the questionnaire, we asked the participants to:

(1) provide the size of the largest feature models they used,
(2) their agreement on several statements about the language (e.g., *it is too complex*),
(3) their general acceptance of the language,
(4) the acceptance for the composition mechanism before and after applying changes,
(5) and their overall preferred language.

Participants were invited to take the questionnaire via the MODEVAR mailing list, reaching many experts from the variability modeling research community. Furthermore, we asked students who were taking courses on software product lines from TU Braunschweig and University of Ulm. The acceptance and understanding of UVL from students is important, as UVL should also be used for teaching and be accessible to non-experts.

Results. Overall, 16 participants submitted their answers to the questionnaire. Figure 7 shows the scores for both variants of UVL with ratings before and after applying their own changes. The colors indicate the given score on a scale from one (very bad, red) to six (very good, green). 14 out of 16 (93.8%) rate the variant using nesting as good (i.e., at least four points). For the variant using references, seven participants give a bad score (i.e., three or lower). Furthermore, references are rated with one or two by overall five

[7] http://doi.org/10.5281/zenodo.5031677
[8] https://github.com/Engelberg/instaparse

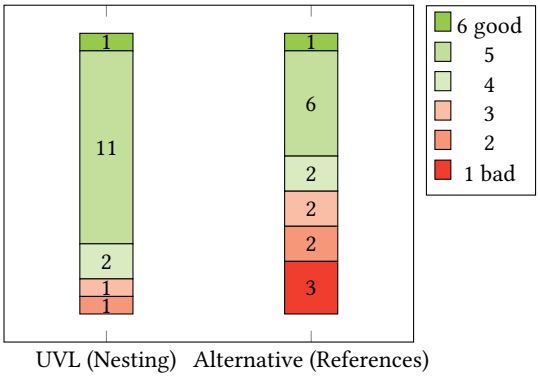

Figure 7: Voting Results for Nesting and References

participants. Every participant that worked with feature models with more than 10,000 features prefers the nesting concept.

The results indicate that the community favors UVL with nesting over the variant using references. In the following, we thus examine the details of the questionnaire only for UVL with nesting.

Figure 8 shows the agreement of the participants for five statements about UVL. Each stacked bar chart corresponds to one statement. The participants could choose between *strongly agree*, *generally agree*, *somewhat disagree*, *strongly disagree*, and *not sure*. The answers given are indicated by the colored blocks in the bar chart (green for agree, red for disagree). 14 out of 16 agree that their feature models could be represented in UVL. Furthermore, only two participants somewhat disagree and none strongly disagrees that UVL can be easily integrated in their tool. Only four (25%) think the language is too complex. In the original survey, the statement was *"It is too complex"*. Due to better readability, we inverted the statement and the answers in the diagram (e.g., strongly disagreeing with the original statement "It is too complex" is here shown as strongly agreeing with "It is not too complex"). There are concerns regarding the style, but 50% of the participants at least generally agree that the style of UVL is good. 11 of 16 participants disagree that UVL is well suited for learning and teaching but only two of those strongly disagree. The qualitative comments on that statement do not give explanations for the comparatively strong disagreement. One reason may be that the participants consider graphical formats to be more suitable for teaching. Nevertheless, the results still indicate a need for further adaptations to UVL regarding its suitability for teaching.

Figure 9 shows the rating of the participants regarding the composition mechanism. The colors indicate the given score on a scale from one (very bad, red) to six (very good, green). 13 of 16 participants rate the mechanism positively (i.e., with at least four points). Only one participant considers the mechanism as very bad.

Discussion. Regarding **RQ2a**, our questionnaire-based survey indicates that UVL with nesting is the preferred variant. Especially participants, who work with large feature models, see the nesting concept more favorable. Furthermore, 15 out of 16 participants stated that they would use the format chosen by the majority instead of their own preference. Thus, we decide to use the variant with nesting for UVL. For **RQ2b**, the results show that UVL is seen

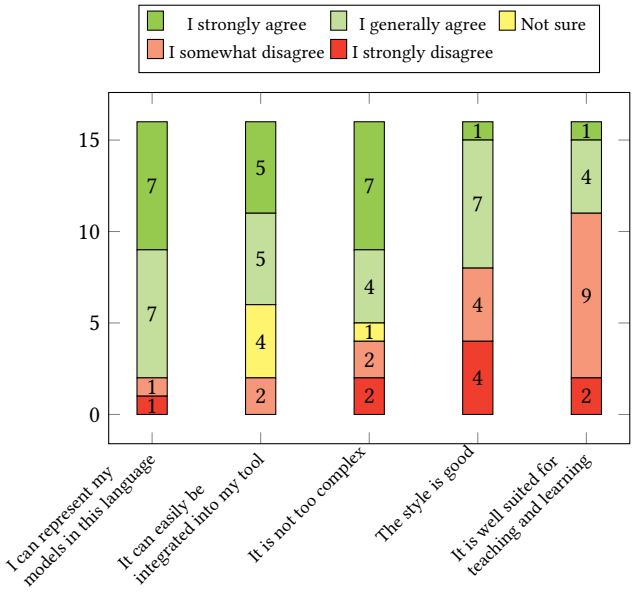

Figure 8: Voting Results on Statements about UVL

Figure 9: Voting Results Composition Mechanism

positively by most participants, which can be used as initial indicator for the acceptance of the community. Nevertheless, there are some concerns in the details (e.g., suitability for teaching) which should be addressed in future versions.

5.3 Real-World Scalability

Another important aspect for the suitability of UVL as a widely accepted notation is its scalability to real-world variability models. We consider two aspects to evaluate the scalability. First, storing a variability model in UVL should not require significantly more size than storing it in another format. Second, the time required to store the variability model should not exceed a second for the vast majority of variability models as saving a model is typically done interactively. Thus, we also evaluate our prototype parser library to examine its runtime. Here, we do not compare the runtimes to the transformations of other formats as each measured runtime is heavily dependent on our implementation and, thus, we would not expect conclusive insights. We use the results regarding size and runtime to answer **RQ3**. We provide the source code, required data, and built scripts to reproduce the following results.[9]

[9] http://doi.org/10.5281/zenodo.5032073

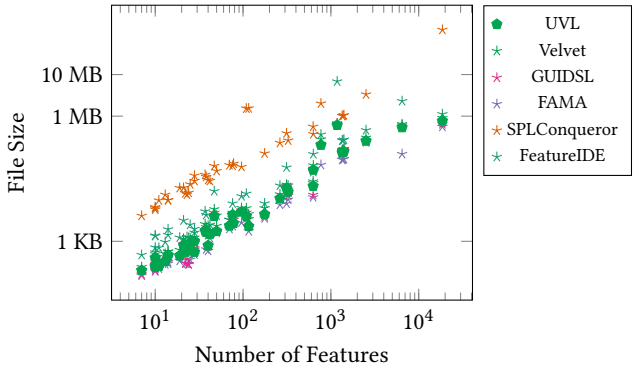

Figure 10: File Sizes of Different Formats

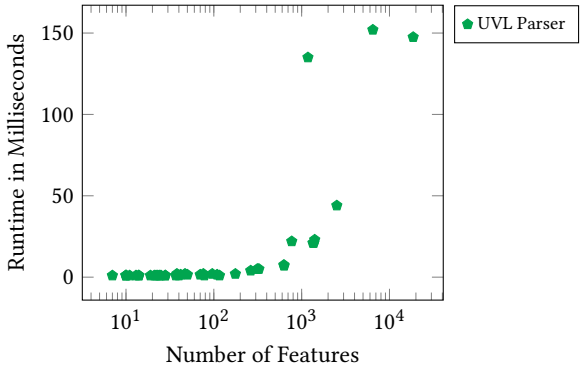

Figure 11: Runtimes of Storing UVL

Experiment Design. We adapted the variability transformation approach TRAVART [32], to transform existing variability models (i.e., feature models, DOPLER decision models [27] and OVM Models [55]) into different feature modeling notations, namely Velvet [59], GUIDSL [9], FAMA [15], SPLConqueror [68], and FeatureIDE XML [44]. Then, we compare file sizes of the different formats. In addition, we performed ten runs for storing each variability model with our UVL parser and measure the runtime.

Subject Systems. Overall, we evaluated UVL on 42 real-world variability models from a variety of domains. The models contain up to 18,617 features and 3,500 constraints. The original models were provided in a variety of formats (e.g., FeatureIDE feature models or DOPLER decision models). We provide each evaluated variability model in the original and UVL format in a GitHub repository[10].

Results. Figure 10 shows the file sizes of all analyzed variability models in the evaluated formats. The x-axis shows the number of features for the corresponding variability models. The y-axis indicates the file size in kilobytes. Each marker corresponds to a format. For each variability model, UVL is among the three formats requiring the least memory. UVL and FAMA are the only formats that require less than 1 MB to store each single considered variability model. For the largest model regarding the file size, FeatureIDE XML requires more than eleven times the space of UVL.

Figure 11 shows the runtimes in milliseconds for storing UVL. In sum, our parser library for UVL required not even a second to store all 42 evaluated variability models. The median of runtimes is 1.5 milliseconds. The largest runtime is 152 milliseconds for Linux.

Discussion. Our inspections show that UVL is among the smallest formats regarding required memory. Furthermore, UVL requires less than 1 MB to store each single evaluated model. Thus, we argue that required memory is an advantage of UVL over many other established formats (e.g., FeatureIDE XML) and no threat to its usability. Our parser library requires less than 0.2 seconds to store each variability model. For **RQ3**, our results strongly indicate that UVL scales to real-world variability models.

5.4 Threats to Validity

Participant Bias. Initially, Engelhardt developed and proposed the language design of UVL and the questionnaires in cooperation with Thüm [30]. During that time, the remaining authors of this work, namely Sundermann, Feichtinger, and Rabiser, were part of at least one of the questionnaires but solely as participants. However, neither of the three authors was involved with developing UVL nor planning to be during this time. Thus, we argue that the given answers can be considered without resulting in a bias as the authors only later joined the initiative. Furthermore, we assume that removing the submissions of participants that are part of the community would increase the bias moreso. In general, we argue that there are several aspects which reduce the effect of particpant bias. Participants could submit their answers anonymously to encourage negative feedback. Also, there were several participants involved in the development of different variability model tools (e.g., FAMA [14], FAMILIAR [3], and FeatureIDE [44]) which should decrease the chance of overfitting for a single tool.

Sample Size. For the first and second questionnaire, there were 20 and 16 participants, respectively, which is a potential threat to validity when transferring the results to the general public. However, we argue that the composition of participants is a reasonable indicator on the acceptance of the community. First, the participants from the MODEVAR initiative are a driving force for a unified variability language. Second, students indicate the acceptance of users not as familiar with variability models and the suitability of UVL for teaching. Nevertheless, more feedback must be gathered, e.g., from further academic experts (especially via this publication) and particularly also from industrial experts. However, our results provide initial evidence that the community could accept our proposal.

Evaluated Formats. We did not evaluate the size of feature models in each available textual format. Every additional format requires to implement a transformation, which induces considerable effort even for a single format. While there may be additional formats that require less space than UVL, in our experiments UVL never required an amount of memory or time for transformation that would have a negative impact on its usability. Thus, evaluating additional formats does not change the conclusion that the scalability of UVL to real-world variability models is no threat to its usability.

Transferring the Scalability. It is possible that the results regarding the scalability of UVL for our variability models may not be transferable to other models. However, we evaluated variability models from a multitude of domains and original formats. Furthermore, our benchmark includes the largest real-world variability models

[10] http://doi.org/10.5281/zenodo.5031829

publicly available. Another potential threat is the computational bias for the runtimes. However, we performed ten runs for each transformation with our UVL parser to vastly reduce its effect.

6 RELATED WORK

We provide an overview on the publications from the MODEVAR initiative which influenced our work and discuss other proposals for unified languages, language surveys, and graphical formats.

MODEVAR. The MODEVAR initiative started at 2019 and aims to find a consensus for modeling variability [13]. As previously stated, our work and the design of UVL is the result of the joint efforts by the community of MODEVAR [10, 11, 17, 31, 35, 42, 51, 58, 75, 77]. In particular, Berger and Collet [17] summarized the application scenarios for variability languages proposed by the community, which is an important aspect to the design of UVL. Ter Beek et al. [11] provide an overview and analysis of existing variability languages. Galindo and Benavides [35] propose requirements for a repository that can be used to exchange feature models which influenced our design decisions regarding our UVL model repository.

Proposals for Unified Languages. A large variety of variability language have been proposed [1, 3, 4, 6, 9, 22, 23, 41, 44, 49, 59, 60, 76]. The Common Variability Language (CVL) aimed to come up with a common meta-model and standard for variability models [37]. In 2017, ISO 26558 was introduced to standardize several aspects of product line development.[11] Villota et al. [77] propose a flexible high-level variability language (HLVL) capable of modeling a large variety of concepts appearing in different variability languages. For the information exchange between variability management tools, Schulze and Hellebrand [64] developed the Variability Exchange Language (VEL). Schobbens et al. [62] presented a generic formal semantics for feature diagrams, based on analyzing existing variants. Other approaches proposed a unified feature language based on comparisons of existing feature meta-models [67], developed a common interface for (temporal) feature models to harmonize feature model notations [39], or provided a an extensible language kernel for variability models [69]. However, neither of the presented proposals was widely adopted by the community. Therefore, there is still a need for a unified language. UVL is the first approach designed in tight cooperation with the community. Thus, we expect that UVL is more promising to be widely adopted. This claim is also supported by the results of our second questionnaire-based survey.

Language Surveys. Several works survey existing variability languages and approaches [8, 12, 21, 25, 28, 36, 57, 61]. Some works focus on comparing concrete types of variability models [24, 29, 66] showing that at least some variability modeling approaches are equivalent, when there is a powerful constraint language available. In our work, we also analyze concrete variability modeling languages to extract the most common structural properties of the languages which we use to evaluate the preferences of community.

Graphical Formats. There are different graphical notations for variability models. However, feature diagrams are widely adopted to visualize feature models and typically only differ in details [3, 14, 15, 44, 78]. A unified textual format could be differently visualized in different tools without requiring additional transformations. Thus, we target only a universal textual notation with UVL.

[11] https://www.iso.org/obp/ui/#iso:std:43117:en

7 CONCLUSION & FUTURE WORK

Researchers and practitioners face a variety of challenges due to the multitude of available variability model formats. A language widely adopted by the community would allow easy access to the datasets and analyses used by other researchers. Furthermore, there are strong indicators that the community desires a unified format. The main contribution of this work was evaluating the preferences of the community regarding the language's scope and its syntax. Using the results, we make an initial attempt for the first variability language that is designed in a joint effort with the community and, in particular, with the MODEVAR initiative. While we provide tooling for UVL, we specifically do not consider the language as final. Instead, we make a first proposal and plan on continually improving and extending UVL in joint work with the community.

Our main goal for the future is to unify the community behind a single language. UVL seems like a promising starting point as our results indicate that the community widely accepts UVL's basic features and its notation. It requires a community effort to assess strength and weaknesses of the proposed language for different applications. That is, the language should be used to teach product lines to students, to build research prototypes, and to evaluate it in industrial applications. The gathered insights can be used to finally provide a unified language.

Currently, UVL allows to represent a large variety of properties (e.g., tool-specific data or a mapping to implementation artifacts) as feature attributes. In the future, it may be beneficial to implement designated mechanisms for such properties. We also consider several extensions for UVL, such as support for first-order-logic formulas, which would allow the use of quantifiers and imposing constraints on feature attributes. While the language proposal is devoted to variability modeling, a dedicated language may also be developed to store configurations.

ACKNOWLEDGMENTS

This work is based on the master's thesis of Dominik Engelhardt. Particular thanks to Ina Schaefer from TU Braunschweig for the helpful feedback on the thesis. We thank David Benavides for the insightful discussion about our ideas and his general commitment to the community. We thank the MODEVAR initiative [12] and especially the organizers Don Batory, Mathieu Acher, and Philippe Collet, and of course David for the continuous effort on designing a unified variability language which highly influenced the design of UVL. Furthermore, we thank all participants of the questionnaires. The financial support by the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology and Development, and the Christian Doppler Research Association is gratefully acknowledged. This work has been partially supported by the German Research Foundation within the project VariantSync (TH 2387/1-1).

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 *Proc. Int'l Workshop on Variability Modelling of Software-Intensive Systems (VaMoS)*. ACM, New York, NY, USA, 101–105.

[12] List of initial supporters: https://modevar.github.io/2019/committees/

[2] Mathieu Acher, Anthony Cleve, Gilles Perrouin, Patrick Heymans, Charles Vanbeneden, Philippe Collet, and Philippe Lahire. 2012. On Extracting Feature Models From Product Descriptions. In *Proc. Int'l Workshop on Variability Modelling of Software-Intensive Systems (VaMoS)* (Leipzig, Germany). ACM, New York, NY, USA, 45–54. https://doi.org/10.1145/2110147.2110153

[3] 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 (SCP)* 78, 6 (2013), 657–681.

[4] Ali Fouad Al-Azzawi. 2018. PyFml-a Textual Language For Feature Modeling. *Int'l J. of Software Engineering & Applications (IJSEA)* 9, 1 (2018), 41–53.

[5] Mustafa Al-Hajjaji, Sebastian Krieter, Thomas Thüm, Malte Lochau, and Gunter Saake. 2016. IncLing: Efficient Product-line Testing Using Incremental Pairwise Sampling. In *Proc. Int'l Conf. on Generative Programming: Concepts & Experiences (GPCE)* (Amsterdam, Netherlands). ACM, New York, NY, USA, 144–155. https://doi.org/10.1145/2993236.2993253

[6] 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 (SQJ)* 27, 1 (2019), 307–347.

[7] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. 2017. *The SMT-LIB Standard: Version 2.6.* Technical Report. Department of Computer Science, The University of Iowa. Available at `www.SMT-LIB.org`.

[8] Rabih Bashroush, Muhammad Garba, Rick Rabiser, Iris Groher, and Goetz Botterweck. 2017. Case Tool Support for Variability Management in Software Product Lines. *ACM Computing Surveys (CSUR)* 50, 1 (2017), 14:1–14:45.

[9] Don Batory. 2005. Feature Models, Grammars, and Propositional Formulas. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC).* Springer, Berlin, Heidelberg, Germany, 7–20.

[10] Don Batory. 2019. Should Future Variability Modeling Languages Express Constraints in OCL?. In *Proc. Int'l Workshop on Languages for Modelling Variability (MODEVAR)* (Paris, France). ACM, New York, NY, USA, 182. https://doi.org/10.1145/3307630.3342406

[11] Maurice H. ter Beek, Klaus Schmid, and Holger Eichelberger. 2019. Textual Variability Modeling Languages: An Overview and Considerations. In *Proc. Int'l Workshop on Languages for Modelling Variability (MODEVAR)* (Paris, France). ACM, New York, NY, USA, 151–157. https://doi.org/10.1145/3307630.3342398

[12] Maurice H. ter Beek, Klaus Schmid, and Holger Eichelberger. 2019. Textual Variability Modeling Languages: An Overview and Considerations. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)* (Paris, France). ACM, New York, NY, USA, 151—-157.

[13] David Benavides, Rick Rabiser, Don Batory, and Mathieu Acher. 2019. First International Workshop on Languages for Modelling Variability (MODEVAR 2019). In *Proc. Int'l Systems and Software Product Line Conf. (SPLC).* ACM, New York, NY, USA, 323–323.

[14] David Benavides, Sergio Segura, and Antonio Ruiz-Cortés. 2010. Automated Analysis of Feature Models 20 Years Later: A Literature Review. *Information Systems* 35, 6 (2010), 615–708.

[15] David Benavides, Sergio Segura, Pablo Trinidad, and Antonio Ruiz-Cortés. 2007. FAMA: Tooling a Framework for the Automated Analysis of Feature Models. In *Proc. Int'l Workshop on Variability Modelling of Software-Intensive Systems (VaMoS)* (Limerick, Ireland). Technical Report 2007-01, Lero, Limerick, Ireland, 129–134.

[16] David Benavides, Pablo Trinidad, and Antonio Ruiz-Cortés. 2005. Using Constraint Programming to Reason on Feature Models. In *Proc. Int'l Conf. on Software Engineering and Knowledge Engineering (SEKE).* 677–682.

[17] Thorsten Berger and Philippe Collet. 2019. Usage Scenarios for a Common Feature Modeling Language. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC).* ACM, New York, NY, USA, 174–181.

[18] 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 Trans. on Software Engineering (TSE)* 39, 12 (2013), 1611–1640. https://doi.org/10.1109/TSE.2013.34

[19] Yves Bontemps, Patrick Heymans, Pierre-Yves Schobbens, and Jean-Christophe Trigaux. 2004. Semantics of Feature Diagrams. In *Proc. Int'l Workshop on Software Variability Management for Product Derivation - Towards Tool Support (SVMPD).*

[20] Tim Bray, Jean Paoli, C Michael Sperberg-McQueen, Eve Maler, François Yergeau, and John Cowan. 2000. Extensible Markup Language (XML) 1.0.

[21] Lianping Chen and Muhammad Ali Babar. 2011. A Systematic Review of Evaluation of Variability Management Approaches in Software Product Lines. *J. Information and Software Technology (IST)* 53, 4 (2011), 344–362.

[22] Dave Clarke, Radu Muschevici, José Proença, Ina Schaefer, and Rudolf Schlatte. 2010. Variability modelling in the ABS language. In *Proc. Int'l Symposium on Formal Methods for Components and Objects (FMCO).* Springer, Cham, Switzerland, 204–224.

[23] Andreas Classen, Quentin Boucher, and Patrick Heymans. 2011. A Text-Based Approach to Feature Modelling: Syntax and Semantics of TVL. *Science of Computer Programming (SCP)* 76, 12 (2011), 1130–1143. Special Issue on Software Evolution, Adaptability and Variability.

[24] Krzysztof Czarnecki, Paul Grünbacher, Rick Rabiser, Klaus Schmid, and Andrzej Wąsowski. 2012. Cool Features and Tough Decisions: A Comparison of Variability Modeling Approaches. In *Proc. Int'l Workshop on Variability Modelling of Software-Intensive Systems (VaMoS)* (Leipzig, Germany). ACM, New York, NY, USA, 173–182. https://doi.org/10.1145/2110147.2110167

[25] Krzysztof Czarnecki, Simon Helsen, and Ulrich Eisenecker. 2005. Formalizing Cardinality-Based Feature Models and Their Specialization. *Software Process: Improvement and Practice* 10 (2005), 7–29.

[26] Krzysztof Czarnecki and Andrzej Wąsowski. 2007. Feature Diagrams and Logics: There and Back Again. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC).* IEEE, Washington, DC, USA, 23–34.

[27] Deepak Dhungana, Paul Grünbacher, and Rick Rabiser. 2011. The DOPLER Meta-Tool for Decision-Oriented Variability Modeling: A Multiple Case Study. *Automated Software Engineering* 18, 1 (2011), 77–114.

[28] Holger Eichelberger and Klaus Schmid. 2015. Mapping the Design Space of Textual Variability Modeling Languages: A Refined Analysis. *Int'l J. Software Tools for Technology Transfer (STTT)* 17, 5 (2015), 559–584. https://doi.org/10.1007/s10009-014-0362-x

[29] 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 *Proc. Int'l Systems and Software Product Line Conf. (SPLC).* ACM, New York, NY, USA, 126–135.

[30] Dominik Engelhardt. 2020. *Towards a Universal Variability Language.* Master's thesis. TU Braunschweig, Germany.

[31] Kevin Feichtinger and Rick Rabiser. 2020. Towards Transforming Variability Models: Usage Scenarios, Required Capabilities and Challenges. In *Proc. Int'l Workshop on Languages for Modelling Variability (MODEVAR)* (Montreal, QC, Canada) *(SPLC '20).* ACM, New York, NY, USA, 44—-51. https://doi.org/10.1145/3382026.3425768

[32] Kevin Feichtinger, Johann Stöbich, Dario Romano, and Rick Rabiser. 2021. TRAVART: An Approach for Transforming Variability Models. In *Proc. Int'l Working Conf. on Variability Modelling of Software-Intensive Systems (VaMoS)* (Krems, Austria). ACM, New York, NY, USA, Article 8, 10 pages. https://doi.org/10.1145/3442391.3442400

[33] Martin Fowler. 2003. *UML Distilled: A Brief Guide to the Standard Object Modeling Language (3rd Edition)* (3 ed.). Addison-Wesley Professional.

[34] Claudia Fritsch, Richard Abt, and Burkhardt Renz. 2020. The Benefits of a Feature Model in Banking. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)* (Montreal, QC, Canada). ACM, New York, NY, USA, Article 9, 11 pages. https://doi.org/10.1145/3382025.3414946

[35] José A. Galindo and David Benavides. 2019. Towards a New Repository for Feature Model Exchange. In *Proc. Int'l Workshop on Languages for Modelling Variability (MODEVAR)* (Paris, France). ACM, New York, NY, USA, 170–173. https://doi.org/10.1145/3307630.3342405

[36] Matthias Galster, Danny Weyns, Dan Tofan, Bartosz Michalik, and Paris Avgeriou. 2013. Variability in Software Systems-a Systematic Literature Review. *IEEE Trans. on Software Engineering (TSE)* 40, 3 (2013), 282–306.

[37] Øystein Haugen, Andrzej Wąsowski, and Krzysztof Czarnecki. 2012. CVL: Common Variability Language. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)* (Salvador, Brazil). ACM, New York, NY, USA, 266–267. https://doi.org/10.1145/2364412.2364462

[38] Marc Hentze, Tobias Pett, Thomas Thüm, and Ina Schaefer. 2021. Hyper Explanations for Feature-Model Defect Analysis. In *Proc. Int'l Working Conf. on Variability Modelling of Software-Intensive Systems (VaMoS)* (Krems, Austria). ACM, New York, NY, USA, Article 14, 9 pages. https://doi.org/10.1145/3442391.3442406

[39] Daniel Hinterreiter, Michael Nieke, Lukas Linsbauer, Christoph Seidl, Herbert Prähofer, and Paul Grünbacher. 2019. Harmonized Temporal Feature Modeling to Uniformly Perform, Track, Analyze, and Replay Software Product Line Evolution. In *Proc. Int'l Conf. on Generative Programming: Concepts & Experiences (GPCE).* ACM, New York, NY, USA, 115–128.

[40] Ayelet Israeli and Dror G. Feitelson. 2010. The Linux Kernel as a Case Study in Software Evolution. *J. Systems and Software (JSS)* 83, 3 (2010), 485–501.

[41] Paulius Juodisius, Atrisha Sarkar, Raghava Rao Mukkamala, Michal Antkiewicz, Krzysztof Czarnecki, and Andrzej Wasowski. 2018. Clafer: Lightweight Modeling of Structure, Behaviour, and Variability. https://arxiv.org/pdf/1807.08576v1. *Computing Research Repository (CoRR)* (2018).

[42] Seiede Reyhane Kamali, Shirin Kasaei, and Roberto E. Lopez-Herrejon. 2019. Answering the Call of the Wild? Thoughts on the Elusive Quest for Ecological Validity in Variability Modeling. In *Proc. Int'l Workshop on Languages for Modelling Variability (MODEVAR)* (Paris, France). ACM, New York, NY, USA, 143–150. https://doi.org/10.1145/3307630.3342400

[43] Gabor Karsai, Holger Krahn, Claas Pinkernell, Bernhard Rumpe, Martin Schindler, and Steven Völkel. 2014. Design Guidelines for Domain Specific Languages. *Computing Research Repository (CoRR)* (2014).

[44] Christian Kästner, Thomas Thüm, Gunter Saake, Janet Feigenspan, Thomas Leich, Fabian Wielgorz, and Sven Apel. 2009. FeatureIDE: A Tool Framework for Feature-Oriented Software Development. In *Proc. Int'l Conf. on Software Engineering (ICSE)* (Vancouver, Canada). IEEE, Washington, DC, USA, 611–614.

Formal demonstration paper.
[45] Sebastian Krieter, Thomas Thüm, Sandro Schulze, Gunter Saake, and Thomas Leich. 2020. YASA: Yet Another Sampling Algorithm. In *Proc. Int'l Working Conf. on Variability Modelling of Software-Intensive Systems (VaMoS)* (Magdeburg, Germany). ACM, New York, NY, USA, Article 4, 10 pages. https://doi.org/10.1145/3377024.3377042
[46] Andreas Kübler, Christoph Zengler, and Wolfgang Küchlin. 2010. Model Counting in Product Configuration. In *Proc. Int'l Workshop on Logics for Component Configuration (LoCoCo)* (Edinburgh, UK). Open Publishing Association, Waterloo, Australia, 44–53. https://doi.org/10.4204/EPTCS.29.5
[47] Jabier Martinez, Wesley K. G. Assunção, and Tewfik Ziadi. 2017. ESPLA: A Catalog of Extractive SPL Adoption Case Studies. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)* (Sevilla, Spain). ACM, New York, NY, USA, 38–41. https://doi.org/10.1145/3109729.3109748
[48] Jens Meinicke, Thomas Thüm, Reimar Schröter, Fabian Benduhn, Thomas Leich, and Gunter Saake. 2017. *Mastering Software Variability with FeatureIDE*. Springer, Berlin, Heidelberg, Germany. https://doi.org/10.1007/978-3-319-61443-4
[49] Marcílio Mendonça, Moises Branco, and Donald Cowan. 2009. S.P.L.O.T.: Software Product Lines Online Tools. In *Proc. Conf. on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA)*. ACM, New York, NY, USA, 761–762.
[50] Terence J. Parr and Russell W. Quong. 1995. ANTLR: A Predicated-LL(k) Parser Generator. *Software: Practice and Experience* 25, 7 (1995), 789–810. https://doi.org/10.1002/spe.4380250705
[51] Pablo Parra, Óscar R. Polo, Segundo Esteban, Agustín Martínez, and Sebastián Sánchez. 2019. A Component-Based Approach to Feature Modelling. In *Proc. Int'l Workshop on Languages for Modelling Variability (MODEVAR)* (Paris, France). ACM, New York, NY, USA, 137––142. https://doi.org/10.1145/3307630.3342402
[52] Gilles Perrouin, Sebastian Oster, Sagar Sen, Jacques Klein, Benoit Baudry, and Yves Le Traon. 2012. Pairwise Testing for Software Product Lines: Comparison of Two Approaches. *Software Quality Journal (SQJ)* 20, 3-4 (2012), 605–643. https://doi.org/10.1007/s11219-011-9160-9
[53] Gilles Perrouin, Sagar Sen, Jacques Klein, Benoit Baudry, and Yves Le Traon. 2010. Automated and Scalable T-Wise Test Case Generation Strategies for Software Product Lines. In *Proc. Int'l Conf. on Software Testing, Verification and Validation (ICST)*. IEEE, Washington, DC, USA, 459–468. https://doi.org/10.1109/ICST.2010.43
[54] Felipe Pezoa, Juan L. Reutter, Fernando Suarez, Martín Ugarte, and Domagoj Vrgoč. 2016. Foundations of JSON Schema. In *Proceedings of the 25th International Conference on World Wide Web* (Montréal, Québec, Canada) *(WWW '16)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 263–273. https://doi.org/10.1145/2872427.2883029
[55] Klaus Pohl, Günter Böckle, and Frank J. van der Linden. 2005. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer, Berlin, Heidelberg, Germany.
[56] Steven David Prestwich. 2009. CNF Encodings. *Handbook of Satisfiability* 185 (2009), 75–97.
[57] Mikko Raatikainen, Juha Tiihonen, and Tomi Männistö. 2019. Software Product Lines and Variability Modeling: A Tertiary Study. *J. Systems and Software (JSS)* 149 (2019), 485–510. https://doi.org/10.1016/j.jss.2018.12.027
[58] Rick Rabiser. 2019. Feature Modeling vs. Decision Modeling: History, Comparison and Perspectives. In *Proc. Int'l Workshop on Languages for Modelling Variability (MODEVAR)* (Paris, France). ACM, New York, NY, USA, 134–136. https://doi.org/10.1145/3307630.3342399
[59] Marko Rosenmüller, Norbert Siegmund, Thomas Thüm, and Gunter Saake. 2011. Multi-Dimensional Variability Modeling. In *Proc. Int'l Workshop on Variability Modelling of Software-Intensive Systems (VaMoS)* (Namur, Belgium). ACM, New York, NY, USA, 11–22.
[60] Klaus Schmid, Christian Kröher, and Sascha El-Sharkawy. 2018. Variability Modeling With the Integrated Variability Modeling Language (IVML) and EASy-Producer. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)*. ACM, New York, NY, USA, 306–306.
[61] Pierre-Yves Schobbens, Patrick Heymans, and Jean-Christophe Trigaux. 2006. Feature Diagrams: A Survey and a Formal Semantics. In *Proc. Int'l Conf. on Requirements Engineering (RE)*. IEEE, Washington, DC, USA, 136–145. https://doi.org/10.1109/RE.2006.23
[62] Pierre-Yves Schobbens, Patrick Heymans, Jean-Christophe Trigaux, and Yves Bontemps. 2007. Generic Semantics of Feature Diagrams. *Computer Networks* 51, 2 (2007), 456–479.
[63] Reimar Schröter, Sebastian Krieter, Thomas Thüm, Fabian Benduhn, and Gunter Saake. 2016. Feature-Model Interfaces: The Highway to Compositional Analyses of Highly-Configurable Systems. In *Proc. Int'l Conf. on Software Engineering (ICSE)* (Austin, Texas). ACM, New York, NY, USA, 667–678. https://doi.org/10.1145/2884781.2884823
[64] Michael Schulze and Robert Hellebrand. 2015. Variability Exchange Language-A Generic Exchange Format for Variability Data. In *Software Engineering (Workshops)*. 71–80.
[65] Sergio Segura, José A. Galindo, David Benavides, José A. Parejo, and Antonio Ruiz-Cortés. 2012. BeTTy: Benchmarking and Testing on the Automated Analysis of Feature Models. In *Proc. Int'l Workshop on Variability Modelling of Software-Intensive Systems (VaMoS)* (Leipzig, Germany). ACM, New York, NY, USA, 63–71. https://doi.org/10.1145/2110147.2110155
[66] Christoph Seidl, Tim Winkelmann, and Ina Schaefer. 2016. A Software Product Line of Feature Modeling Notations and Cross-Tree Constraint Languages. In *Proc. Modellierung*, Andreas Oberweis and Ralf H. Reussner (Eds.). Gesellschaft für Informatik, Bonn, Germany, 157–172.
[67] S. Sepúlveda, C. Cares, and C. Cachero. 2012. Towards a Unified Feature Metamodel: A Systematic Comparison of Feature Languages. In *Proc. Iberian Conf. on Information Systems and Technologies (CISTI)*. IEEE, Washington, DC, USA, 1–7.
[68] Norbert Siegmund, Marko Rosenmüller, Martin Kuhlemann, Christian Kästner, Sven Apel, and Gunter Saake. 2012. SPL Conqueror: Toward Optimization of Non-functional Properties in Software Product Lines. *Software Quality Journal (SQJ)* 20, 3-4 (Sept. 2012), 487–517. https://doi.org/10.1007/s11219-011-9152-9
[69] Stefan Sobernig and Olaf Lessenich. 2021. V1e: A Kernel for Domain-Specific Textual Variability Modelling Languages. In *Proc. Int'l Working Conf. on Variability Modelling of Software-Intensive Systems (VaMoS)* (Krems, Austria). ACM, New York, NY, USA, Article 4, 7 pages. https://doi.org/10.1145/3442391.3442396
[70] Joshua Sprey, Chico Sundermann, Sebastian Krieter, Michael Nieke, Jacopo Mauro, Thomas Thüm, and Ina Schaefer. 2020. SMT-Based Variability Analyses in FeatureIDE. In *Proc. Int'l Working Conf. on Variability Modelling of Software-Intensive Systems (VaMoS)* (Magdeburg, Germany). ACM, New York, NY, USA, Article 6, 9 pages. https://doi.org/10.1145/3377024.3377036
[71] Chico Sundermann, Tobias Heß, Dominik Engelhardt, Rahel Arens, Johannes Herschel, Kevin Jedelhauser, Benedikt Jutz, Sebastian Krieter, and Ina Schaefer. 2021. Integration of UVL in FeatureIDE. In *Proc. Int'l Workshop on Languages for Modelling Variability (MODEVAR) (SPLC '21)*. ACM, New York, NY, USA. To appear.
[72] Chico Sundermann, Thomas Thüm, and Ina Schaefer. 2020. Evaluating #SAT Solvers on Industrial Feature Models. In *Proc. Int'l Working Conf. on Variability Modelling of Software-Intensive Systems (VaMoS)* (Magdeburg, Germany). ACM, New York, NY, USA, Article 3, 9 pages. https://doi.org/10.1145/3377024.3377025
[73] Maurice H. ter Beek, Alessandro Fantechi, Stefania Gnesi, and Franco Mazzanti. 2016. Modelling and Analysing Variability in Product Families: Model Checking of Modal Transition Systems with Variability Constraints. *J. Logic and Algebraic Methods in Programming (JLAMP)* 85, 2 (2016), 287–315. https://doi.org/10.1016/j.jlamp.2015.11.006
[74] Maurice H. ter Beek, Franco Mazzanti, and Aldi Sulova. 2012. VMC: A Tool for Product Variability Analysis. In *Proc. Int'l Symposium on Formal Methods (FM)* (Paris, France). Springer, Berlin, Heidelberg, Germany, 450–454.
[75] Thomas Thüm, Christoph Seidl, and Ina Schaefer. 2019. On Language Levels for Feature Modeling Notations. In *Proc. Int'l Workshop on Languages for Modelling Variability (MODEVAR)* (Paris, France). ACM, New York, NY, USA, 158–161. https://doi.org/10.1145/3307630.3342404
[76] Arie van Deursen and Paul Klint. 2002. Domain-Specific Language Design Requires Feature Descriptions. *Computing and Information Technology* 10, 1 (2002), 1–17.
[77] Angela Villota, Raúl Mazo, and Camille Salinesi. 2019. The High-Level Variability Language: An Ontological Approach. In *Proc. Int'l Workshop on Languages for Modelling Variability (MODEVAR)* (Paris, France). ACM, New York, NY, USA, 162–169. https://doi.org/10.1145/3307630.3342401
[78] Jules White, David Benavides, Douglas C Schmidt, Pablo Trinidad, Brian Dougherty, and Antonio Ruiz-Cortés. 2010. Automated Diagnosis of Feature Model Configurations. *J. Systems and Software (JSS)* 83, 7 (2010), 1094–1107.