Integration of UVL in FeatureIDE

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

Tobias Heß
University of Ulm
Ulm, Germany
tobias.hess@uni-ulm.de

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

Rahel Arens
University of Ulm
Ulm, Germany
rahel.arens@uni-ulm.de

Johannes Herschel
University of Ulm
Ulm, Germany
johannes.herschel@uni-ulm.de

Kevin Jedelhauser
University of Ulm
Ulm, Germany
kevin.jedelhauser@uni-ulm.de

Benedikt Jutz
University of Ulm
Ulm, Germany
benedikt.jutz@uni-ulm.de

Sebastian Krieter
Harz University of Applied Sciences
Wernigerode, Germany
skrieter@hs-harz.de

Ina Schaefer
TU Braunschweig
Braunschweig, Germany
i.schaefer@tu-braunschweig.de

ABSTRACT

Variability models are prevalent for specifying the commonalities and variabilities of configurable systems. A large variety of tools support using, editing, and analyzing variability models. However, the different tools often depend on distinct textual notations to store and read variability models which induces a large effort for researchers and practitioners. This additional effort could be reduced if the community adopted a single format. Following the goal of the MODEVAR initiative to develop a widely adopted variability language, we provided a first proposal with the Universal Variability language (UVL) in previous work. For a textual format to be adopted, an important aspect is an as small as possible effort when integrating the format in other tools. In this work, we discuss the integration of UVL in FeatureIDE. We use the integration to examine the applicability of UVL and our parser library to existing tools and gather further requirements for the language design. Furthermore, we provide a thorough documentation on the implementation to be used as reference and guidance for integration in other tools.

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, Tobias Heß, Dominik Engelhardt, Rahel Arens, Johannes Herschel, Kevin Jedelhauser, Benedikt Jutz, Sebastian Krieter, and Ina Schaefer. 2021. Integration of UVL in FeatureIDE. In *25th ACM International Systems and Software Product Line Conference - Volume B (SPLC '21), September 6–11, 2021, Leicester, United Kingdom.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3461002.3473940

1 INTRODUCTION

Product lines are widely used to describe families of products. Commonly, such a product line is specified using a variability model (e.g., a feature model or decision model) which defines the commonalities and variabilities of the products [10, 12, 42]. While variability models are widely adopted to describe product lines, the literature considers many different textual formats [1–5, 7, 14–17, 22, 25, 27, 32, 35, 36, 38, 39, 45]. In addition, existing tools typically use different formats to store variability models [2, 11, 27, 32, 47]. Some analyses or models are only available in specific tools or formats. Thus, it is often necessary to implement transformation tooling between different formats to access the analyses or models which induces additional effort for researchers and practitioners. Existing proposals for standardization, such as ISO-43117 [1] or the Common Variability Language (CVL) [23], have not been widely adopted by the community. Furthermore, there are several indicators that the community desires a unified format [6, 8, 9, 13, 19, 21, 26, 34, 37, 44, 46].

In previous work [41], we proposed the Universal Variability Language (UVL) which is the first textual format for variability modeling designed in a joined effort within the community [9]. For the initial language design, we (1) gathered requirements of the community, (2) analyzed existing variability languages to identify popular concepts and (3) evaluated the acceptance of the community [41]. Our observations indicate that the proposal is a promising starting point for a widely adopted variability language.

In addition to the acceptance of the community for the language design, the possibility of integrating the format into variability modeling tools is vital [13]. To this end, we implemented a parser library[2] in prior work [41], which can be used standalone or embedded into other tools, that allows importing and exporting UVL.

[1]https://www.iso.org/obp/ui/#iso:std:43117:en
[2]https://github.com/Universal-Variability-Language/uvl-parser

Prior to this work, UVL was already integrated in the transformation tool TRAVART and rudimentarily in the feature-modeling tool FeatureIDE (since version 3.7.0).

In this work, we provide the first documentation for the integration of UVL in FeatureIDE including new enhancements and improvements that will be released in FeatureIDEv3.8. In particular, we describe the composition mechanism which comes as a feature of UVL and discuss in detail how to use UVL's native language features to represent feature models and also tool-specific data for FeatureIDE. Based on our insights from the integration in FeatureIDE, we analyze strengths and weaknesses of the current version of UVL regarding the applicability to existing tools.

This work is structured as follows. In Section 2, we describe the development, syntax, and features of UVL. In Section 3, we shortly present FeatureIDE, describe the integration of UVL in FeatureIDE, discuss the strength and weaknesses of UVL when integrated into other feature-modeling tools, and analyze limitations to our observations. In Section 4, we conclude on our observations regarding UVL and its integration in FeatureIDE.

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
```

2 UNIVERSAL VARIABILTIY LANGUAGE

The Universal Variability Language (UVL) is primarily developed for exchange between variability modeling tools. However, the scope of the language also includes other requirements specified by the community [13], such as being human editable and suitable for teaching. UVL's design is a joined effort of the community and in particular with the MODEVAR initiative [9]. In Section 2.1, we shortly describe the development process. In Section 2.2, we present the current version of UVL and its syntax. For more details on the development and design of UVL, we refer to our previous work [41].

2.1 Development

Within the MODEVAR initiative, several researchers contributed to the goal of a unified textual format for feature models [6, 8, 13, 19, 21, 26, 34, 37, 44, 46]. In addition to their insights, we evaluated the requirements and preferences of the community to design a language tailored to the community [41].

In the first step, we analyzed existing variability languages to identify concepts that are popular and familiar to users of feature modeling languages. Based on the identified concepts, we performed a questionnaire to evaluate the preferences of the community. For the majority of decisions, the participants favored (with at least 60% of the votes) one option or a combination of options. To evaluate the disputed concepts, we created two variants for further evaluation mainly differing in their representations of hierarchy (i.e., using references or nesting).

In the second step, we performed another questionnaire to evaluate the community's acceptance on our proposal and select the favorably seen variant. The results indicated that the variant using nesting for hierarchy is seen clearly favorable which we thus use as first version of UVL. In particular, each participant rated both versions one a scale from one (very bad) to six (very good) which resulted in an average score of 4.62 for nesting and 3.56 for references. Furthermore, UVL was widely accepted by the participants of the questionnaire (i.e., 87.5% gave a score of at least four).

2.2 Syntax

Listing 1 shows a feature model of a simplified server in UVL. Keywords are highlighted in bold font. The *hierarchy* of the feature model is specified using nesting declared using indentation. Each `Server` requires a `FileSystem` and an `OperatingSystem` (denoted by MANDATORY-group) and may have `Logging` (OPTIONAL). At least one child of `FileSystem` needs to be selected (OR-group). In addition to the supported groups, namely ALTERNATIVE, OR, OPTIONAL, and MANDATORY, UVL supports cardinalities. For example, the cardinality [3..6] denotes that at least three and at most six child features need to be selected. Some features do not correspond to an implementation artifact (e.g., `Server`) which can be indicated with an `abstract` flag. For each feature, additional information can be stored in terms of feature attributes (e.g., the log level of the feature `Logging`). Cross-tree constraints (e.g., `Windows` ⇒ `NTFS`) are stored separately from the tree hierarchy.

Listing 2 shows the composition mechanism of UVL which allows to edit subsets of the feature model in separate files (as seen in Listing 3) and then just reference the submodel in the original model. The referenced submodels are included using a Java-like import system (e.g., `imports OperatingSystem`). Furthermore, each submodel can be given an alias (e.g., `os`). Files that are stored in subdirectories can be accessed similarly to Java packages. The directories are separated with dots (e.g., `submodels.FileSystem`). Features from the submodel are then referenced by submodels name or its alias (e.g., `OperatingSystem.Windows` or `os.Windows`). This syntax is used both in the tree hierarchy and cross-tree constraints. Cross-tree constraints over features of a single submodel can be either denoted in the submodel or the composed feature model.

Listing 4 shows the context-free grammar used for parsing UVL in an EBNF-notation. On the highest abstraction level, each feature model may consist of (1) declaration of its namespace, (2) imports of submodels, (3) a tree-hierarchy, and (4) cross-tree constraints.

The namespace always consists of `namespace`-keyword plus an identifier. Imports are introduced with an `imports`-keyword and consist of an identifier and optionally an alias (i.e., `as <id>`).

Listing 2: Composition Mechanism in UVL

```
namespace Server

imports
  OperatingSystem as os
  submodels.FileSystem as fs

features
  Server {abstract}
    mandatory
      fs.FileSystem
      os.OperatingSystem

  optional
    Logging {
      default,
      log_level "warn" // Feature Attribute
    }

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

Listing 3: Operating System Submodel in UVL

```
namespace OperatingSystem

features
  OperatingSystem {abstract}
    alternative
      Windows
      macOS
      Debian
```

The feature tree is introduced with the `features`-keyword. Each feature consists of an identifier and a set of child nodes which may be either feature attributes or groups. A feature attribute is a key value pair. The key is always a string identifier and the value can be either Boolean, numeric, a string, a set of attributes, a vector, or a constraint. Groups can be `or` (at least one selected), `alternative` (exactly one), `mandatory` (all), `optional` (any number), or a cardinality. A cardinality always has the form `[a..b]` where `a` denotes the minimal number of selected features as an integer and `b` the maximal number either as integer or a wildcard (i.e., `*`). Each group is followed by a, possibly empty, set of features.

Cross-tree constraints are preceded with the `constraints` keyword. Each constraint conforms to propositional logic using the symbols: <=>, =>, | (for $\vee$), & (for $\wedge$), ! (for not), and brackets. Constructs that exceed the expressiveness of propositional logic are not supported in the current version of UVL.

2.3 Parser Library

UVL comes with its own default library that allows importing and exporting variability models in UVL. We provide the parser library at a GitHub-repository with source code, instructions to build, documentation, and examples (Java usage and UVL models).[3] The library is written in the programming language Clojure [24] and uses *instaparse*[4] to handle the CFG shown in Listing 4 to parse UVL. There were suitable alternatives to *instaparse* (e.g., ANTLR [33] and Bison [30]). We decided to use *instaparse* due to its simplicity for

[3]https://github.com/Universal-Variability-Language/uvl-parser
[4]https://github.com/Engelberg/instaparse

Listing 4: UVL Grammar in EBNF Notation

```
FeatureModel = Ns? Imports? Features? Constraints?

Ns = <'namespace'> REF
Imports = <'imports'> (<indent> Import+ <dedent>)?
Import = REF (<'as'> ID)? (<'refer'> Refer)?
Refer = (<'['> (ID <','?>)* <']'>) | 'all'

Features = <'features'> Children?
<Children> = <indent> FeatureSpec+ <dedent>
FeatureSpec = REF Attributes? Groups?
Attributes = (<'{'> <'}'>) |
             (<'{'> Attribute (<','> Attribute)* <'}'>)
Attribute = Key Value?
Key = ID
Value = Boolean|Number|String|Attributes|Vector|Constraint
Boolean = 'true' | 'false'
Number = #'[+-]?(0|[1-9]\d*)(\.\d*)?([eE][+-]?\d+)?'
String = #'"(?:[^"\\\n]|\\.)*"'
Vector = <'['> (Value <','?>)* <']'>
Groups = <indent> Group* <dedent>
Group = ('or' | 'alternative' | 'mandatory' |
           'optional' | Cardinality)
Children?
Cardinality = <'['> (int <'..'>)? (int|'*') <']'>

Constraints = <'constraints'> (<indent> Constraint+ <dedent>)?
<Constraint> = disj-impl | Equiv
Equiv = Constraint <'<=>'> disj-impl
<disj-impl> = disj | Impl
Impl = disj-impl <'=>'> disj
<disj> = conj | Or
Or = disj <'|'> conj
<conj> = term-not | And
And = conj <'&'> term-not
<term-not> = term | Not
Not = <'!'> term
<term> = REF | <'('> Constraint <')'>

indent = '_INDENT_'
dedent = '_DEDENT_'
<strictID> = #'(?!alternative|or|features|constraints|
true|false|as|refer)[a-zA-Z][a-zA-Z_0-9]*'
<ID> = #'(?!true|false)[a-zA-Z][a-zA-Z_0-9]*'
REF = (ID <'.'>)* strictID
<int> = #'0|[1-9]\d*'
```

implementing a parser and as it is written in Clojure which allows to generate a JAR for usage in Java. This should reduce the effort for integrating UVL to various variability modeling tools as most of them use Java [2, 11, 27, 32].

Figure 1 shows the abstract syntax tree (AST) for an `UVLModel` returned by the parser library. Each `UVLModel` consists of potentially empty set of constraints, imports, and features. A constraint is instance of either `And`, `Or`, `Impl`, `Equiv`, `Not`, or a literal described as `String`. Depending on the type, a constraint may reference two (e.g., `And`) or one (`Not`) constraint. An `Import` is stored using two `String`s, namely namespace and alias. A `Feature` has a name, a key-value map of attributes and a set of `Group`-objects. Each `Group` contains a `String` containing its type and integers to lower and upper bounds for cardinality groups. If a parsed file violates the rules of the CFG shown in Listing 4, the parser library instead returns a `ParseError` object which indicates the location of the error (with integers for line and column), an error message, and a list of expected symbols. For each presented object, the fields can be accessed with respective getters and setters. An existing textually specified UVL model can be parsed using `UVLParser`'s method `parse(String)`. An `UVLModel` can be exported using its `toString()` method. For examples on the usage of our parser library in Java, we refer to the provided GitHub repository (see footnote 3).

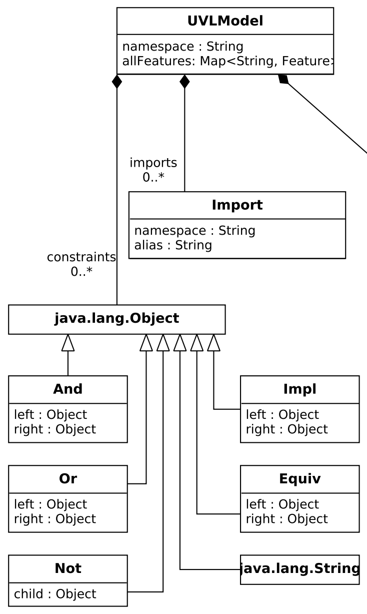

Figure 1: UVLModel AST adapted from [18]

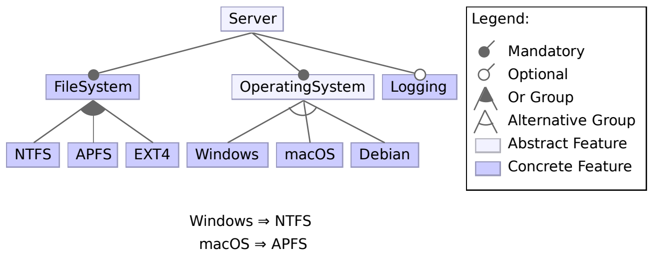

Figure 2: Operating System Submodel in FeatureIDE

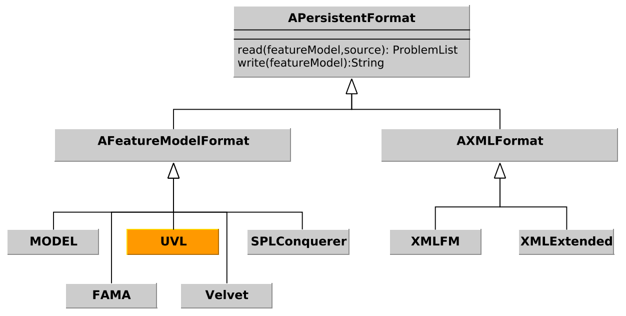

Figure 3: Textual Formats in FeatureIDE

3 INTEGRATION INTO FEATUREIDE

The main purpose of UVL is to be used for exchange between different tools. Thus, we consider the possibility of integrating a variability language into existing tools as vital. Thus, we integrated UVL in the popular tool FeatureIDE [31] to gather insights on the applicability of UVL's features to existing variability-modeling tools. In the following, we (1) give a short overview on FeatureIDE, (2) present the integration of UVL and the newly introduced features, and (3) discuss the observations regarding strengths and possible improvements for UVL. The presented changes will be fully released in FeatureIDEv3.8.[5]

3.1 FeatureIDE

FeatureIDE is an eclipse-based tool for feature-oriented (software) development and supports, amongst other functionalities, editing and analyzing feature models [31]. FeatureIDE can be considered a de facto standard and is widely used in research and industry [19, 20, 28, 29]. Thus, we decided to integrate UVL in FeatureIDE.

Figure 2 shows the feature model of our running example from Listing 1 in FeatureIDE. The tool supports the groups ALTERNATIVE, OR, OPTIONAL, and MANDATORY but no cardinalities. Furthermore, a feature can be either concrete or abstract (i.e., correspond to an implementation artifact or not, respectively).

Currently, FeatureIDE uses an XML-format to store feature models and tool-specific data. However, FeatureIDE offers an interface to integrate other languages. Figure 3 shows the different feature model formats implemented in FeatureIDE. Each of them inherits either from `AFeatureModelFormat` or `AXMLFormat`. Furthermore, each format needs to implement the methods `read()` and `write()` to support importing and exporting feature models, respectively. Formats can also be read-only or write-only. In this case, only one of the `read()` and `write()` methods needs to be implemented. For `FAMA` and `Conquerer`, only writing is supported in FeatureIDE, for instance.

[5]https://github.com/FeatureIDE/FeatureIDE/releases/tag/v3.8.0

3.2 Integration of UVL

Currently, UVL is integrated as one of multiple alternative formats as shown in Figure 3 to FeatureIDE's standard format in XML. To integrate UVL, we implemented support methods to parse features, groups, attributes, and constraints which mainly use the API of our parser library to translate the read feature model to the data structure used in FeatureIDE. As it comes, FeatureIDE now allows to read and write feature models in the UVL format. Furthermore, a feature model in UVL can be translated to each format supported by FeatureIDE and vice versa.

Syntactical errors (i.e., if the feature model does not comfort to the CFG shown in Listing 4) can be detected using the parser library. If an error occurs, the library returns a list of problems including an explanation and the location (line and column) where the error was detected. Afterwards, FeatureIDE uses the returned UVL model to check for semantical errors, such as feature names that appear in constraints but are missing in the tree. Both, syntactical and semantical errors, will prevent the creation of a feature model and instead an error message is displayed to the user.

UVL introduces a composition mechanism to FeatureIDE which allows to reference feature models from different files. With that mechanism, feature models can be decomposed into smaller units which are typically easier to edit. Figure 4 shows a composed feature model in FeatureIDE. Features that are part of a referenced

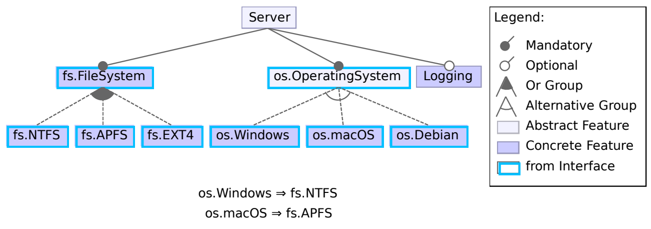

Figure 4: Composition Mechanism in FeatureIDE

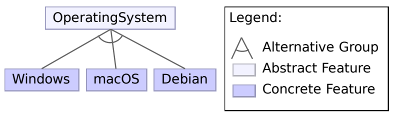

Figure 5: Operating System Submodel in FeatureIDE

submodel are marked with a skyblue border (*from interface* in legend). In addition, the features have a prefix of the provided alias (e.g., `os` or `fs`). Those are also used in the cross-tree constraints (e.g., `os.Windows` ⇒ `fs.NTFS`). Features from a referenced submodel cannot be edited in the composed feature model but only in the respective submodel. Figure 5 shows the submodel of the `OperatingSystem` as an example. Here, the features are editable.

UVL also fully supports the implementation of feature attributes in FeatureIDE. Each type of attributes supported by FeatureIDE, namely long, double, string, and boolean [40], can be read from and stored in the UVL format and used in the attributes view.

For the current version of FeatureIDE, the propositional constraints of UVL are sufficient. However, FeatureIDE plans to support constraints over attributes including numerical constraints in the future. After the introduction of such constraints, UVL would also require an extension with more expressive constraints.

Due to the flexibility of attributes in UVL, the tool-specific data stored in FeatureIDE can also be included in UVL. However, in FeatureIDE, some stored properties correspond to the entire feature model. Such properties need to be attached to features (e.g., the root) in UVL as UVL only allows to attach attributes to specific features.

3.3 Insights & Remarks

The expressiveness of UVL is sufficient to specify each feature model in FeatureIDE. Our parser library in Clojure allows direct access to the API from variability-modeling tools in Java. Each method to parse the different elements in a UVL feature model can be used in the FeatureIDE format wrapper. Overall, the integration in FeatureIDE consists of a single class with less than 500 lines of code. With UVL's integration in FeatureIDE, it is now possible to translate UVL to each format shown in Figure 3 and vice versa. UVL is the first format to support composition of several models in FeatureIDE. Thus, the introduction of UVL allowed our implementation of a composition mechanism in FeatureIDE.

UVL is lacking a feature for storing properties that are not tied to a specific feature. As a workaround, such properties can be attached as UVL attributes to specific features (e.g., the root). Thus, all properties required for FeatureIDE can be stored in an UVL model. However, storing them as generic attributes may cause conflicts with the feature attributes in FeatureIDE and also other tools. Thus, a dedicated mechanism for properties that correspond to the entire feature model may be a beneficial extension for UVL.

Further extensions to FeatureIDE may require extending UVL. While UVL expressiveness is sufficient for current FeatureIDE models, adaptations on FeatureIDE's side (e.g., constraints over attributes) may require updates for UVL.

A feature model stored in UVL may not be supported by FeatureIDE. UVL supports cardinalities, such as [3..6] (i.e., between three and six features need to be selected), to specify parent-children relationships. In contrast, FeatureIDE only supports groups (i.e., alternative, or, mandatory, and optional) and, thus, cannot model cardinalities. Furthermore, the attribute system of UVL is more flexible than feature attributes in FeatureIDE (e.g., UVL supports lists of attributes). Currently, if FeatureIDE reads a UVL model containing syntax elements that are not supported, FeatureIDE provides an error message and does not generate a feature model. To prevent version or functionality mismatches in the future, it may be beneficial to integrate properties indicating the version and language level for UVL. Storing such indicators could be realized by the aforementioned mechanism to store properties for the entire feature model. Also, it may be beneficial to implement translations between different language levels (e.g., transform cardinalities to elements that can be modeled in FeatureIDE).

3.4 Limitations

UVL was designed in cooperation with Thomas Thüm who is part of the FeatureIDE development team [31, 43]. It is reasonable to assume that the design is thus influenced by the design of FeatureIDE. While UVL covers the different concepts required for feature modeling in FeatureIDE, there may be less coverage for other tools. Nevertheless, we argue that FeatureIDE is a reasonable candidate for a first implementation due to its popularity. Furthermore, UVL is more expressive than FeatureIDE's standard format and thus more likely to cover the requirements of other tools.

The parser library in Clojure allows creating a JAR which mainly simplifies access from Java. Thus, the integration in other feature-modeling tools that are not Java-based may require additional effort compared to the integration in FeatureIDE. We cannot assess that additional effort as we only integrated UVL in Java-based tool (FeatureIDE) which can directly access the API of our parser library. However, due to the popularity of Java in variability-modeling tools, Clojure (and thus compiling into JAR) seems a reasonable option for a first parser library. Nevertheless, in the future, further libraries in other programming languages may be beneficial to simplify integration for more tools.

4 CONCLUSION

With the current implementation, UVL is an usable alternative to the standard XML format of FeatureIDE. In addition, UVL comes with supports for the composition mechanism which cannot be used with the standard format. Our parser library in Clojure allowed an integration of the UVL format in FeatureIDE with a single class with less than 1000 lines of code. Nevertheless, we also found some new requirements during the integration in FeatureIDE (e.g., a designated mechanism for storing tool-specific data). While propositional constraints are sufficient for FeatureIDE at the moment, the developers plan to integrate constraints over feature attributes in the future. In this case, UVL would require an extension with more expressive (e.g., first-order logic) constraints. In general, we expect that UVL requires different language levels to be suitable for multiple tools. Thus, in the future an extension mechanism to enable varying language levels (e.g., different expressiveness of constraints) may be beneficial.

ACKNOWLEDGMENTS

In their student's project, Arens, Herschel, Jedelhauser, and Jutz made various enhancements and also fixes on the implementation of UVL in FeatureIDE. This work is build upon on the master's thesis of Dominik Engelhardt [18]. Particular thanks to Thomas Thüm from University of Ulm 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 [6] 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. 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.

[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 (SCP)* 78, 6 (2013), 657–681.

[3] 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.

[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 (SQJ)* 27, 1 (2019), 307–347.

[5] 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.

[6] 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

[7] 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.

[8] 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

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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

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

[19] 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

[20] 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

[21] 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

[22] 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.

[23] 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

[24] Rich Hickey. 2008. The Clojure Programming Language. In *Proceedings of the 2008 Symposium on Dynamic Languages* (Paphos, Cyprus) *(DLS '08)*. ACM, New York, NY, USA, Article 1, 1 pages. https://doi.org/10.1145/1408681.1408682

[25] 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).

[26] 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

[27] 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.

[28] Matthias Kowal, Sofia Ananieva, and Thomas Thüm. 2016. Explaining Anomalies in Feature Models. In *Proc. Int'l Conf. on Generative Programming: Concepts & Experiences (GPCE)* (Amsterdam, Netherlands). ACM, New York, NY, USA, 132–143. https://doi.org/10.1145/2993236.2993248

[29] 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

[30] John Levine. 2009. *Flex & Bison: Text Processing Tools.* "O'Reilly Media, Inc.".

[31] 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

[32] 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.

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

[33] 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

[34] 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

[35] Klaus Pohl, Günter Böckle, and Frank J. van der Linden. 2005. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer, Berlin, Heidelberg, Germany.

[36] 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

[37] 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

[38] 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.

[39] 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.

[40] 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

[41] 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 *Proc. Int'l Systems and Software Product Line Conf. (SPLC)*. ACM, New York, NY, USA. To appear.

[42] Thomas Thüm, Don Batory, and Christian Kästner. 2009. Reasoning About Edits to Feature Models. In *Proc. Int'l Conf. on Software Engineering (ICSE)* (Vancouver, Canada). IEEE, Washington, DC, USA, 254–264. https://doi.org/10.1109/ICSE.2009.5070526

[43] Thomas Thüm, Christian Kästner, Fabian Benduhn, Jens Meinicke, Gunter Saake, and Thomas Leich. 2014. FeatureIDE: An Extensible Framework for Feature-Oriented Software Development. *Science of Computer Programming (SCP)* 79, 0 (Jan. 2014), 70–85. https://doi.org/10.1016/j.scico.2012.06.002

[44] 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

[45] Arie van Deursen and Paul Klint. 2002. Domain-Specific Language Design Requires Feature Descriptions. *Computing and Information Technology* 10, 1 (2002), 1–17.

[46] 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

[47] 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.