UVLParser: Extending UVL with Language Levels and Conversion Strategies

Chico Sundermann
University of Ulm
Germany

Stefan Vill
University of Ulm
Germany

Thomas Thüm
University of Ulm
Germany

Kevin Feichtinger
LIT CPS Lab
Johannes Kepler University Linz
Austria

Prankur Agarwal
CDL VaSiCS, LIT CPS Lab
Johannes Kepler University Linz
Austria

Rick Rabiser
CDL VaSiCS, LIT CPS Lab
Johannes Kepler University Linz
Austria

José A. Galindo
University of Seville
Spain

David Benavides
University of Seville
Spain

ABSTRACT

The Universal Variability Language (UVL) is a community effort towards a widely adopted textual specification for feature models. For widespread usage, the language should be simple to understand and easy to embed in existing tools. Also, many different use cases should be covered, which requires an expressive language design. To incorporate these clashing requirements in UVL, we enrich the language with several optional extensions that add more expressive language features. Furthermore, we provide conversion strategies that translate between those language levels by replacing the complex constructs with equivalent but simpler ones. With our library, other tool developers can select their supported language levels and automatically convert more complex language constructs. Those constructs are then replaced with semantically equivalent expressions that are supported by the tool.

CCS CONCEPTS

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

KEYWORDS

feature model, variability model, variability language

ACM Reference Format:
Chico Sundermann, Stefan Vill, Thomas Thüm, Kevin Feichtinger, Prankur Agarwal, Rick Rabiser, José A. Galindo, and David Benavides. 2023. UVLParser: Extending UVL with Language Levels and Conversion Strategies. In *27th ACM International Systems and Software Product Line Conference - Volume B (SPLC '23), August 28-September 1, 2023, Tokyo, Japan.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3579028.3609013

SPLC '23, August 28-September 1, 2023, Tokyo, Japan

ACM ISBN 979-8-4007-0092-7/23/08.
https://doi.org/10.1145/3579028.3609013

1 INTRODUCTION

The Universal Variability Language (UVL) is textual notation for variability models recently developed as community effort [16]. With UVL, the MODEVAR initiative [2] aims to achieve widespread adoption for one language as the multitude of available formats hinders exchange between tools.

With the variety of use cases in practice, different language features are often demanded. For example, numerical constraints over feature attributes (e.g., limiting the amount of power consumption) are often considered in variability languages [1]. Still, some popular formats, such as FeatureIDE xml [12], do not provide support for numerical constraints.

We extend UVL with optional language levels to enable a tailored language specification depending on the use case. In particular, we provide a first concrete realization of earlier suggestions on language levels for feature models [17]. With the language levels, UVL still has a simple core language but is able to cover more sophisticated expressions. The language levels encapsulate constructs that can be potentially analyzed with the same reasoning engine [17]. We envision that variability modeling tools can then select the supported language levels based on the underlying reasoning engine. For instance, FeatureIDE uses a SAT solver (SAT4J [11]) as backend and, thus, should use the *Boolean* level of UVL. Clafer [10] uses SMT [6] and, thus, could also use the *Arithmetic* language level.

To simplify exchange, we extend the notion of language levels [17] by adding conversion strategies between those levels. Simple exchange of models between tools is vital for the usability [3]. Employing the language levels without further support vastly hinders exchange as UVL models specified in more complex language levels are not directly usable in tools that only support simpler levels. A conversion strategy can be used to automatically transform complex language levels in simpler constructs that are supported by the tool of interest.

Our Java-based library `UVLParser` extends UVL with two new (resulting in overall three) *major language levels* that each encapsulate related language features. In particular, the three major levels contain (1) *boolean* expressions over features, (2) *arithmetic* expressions over features and attributes, and (3) features with a non-boolean *type* (e.g., integer features). Each major level consists

of some core language features and small optional extensions (i.e., *minor levels*). `UVLParser` supports reading, writing, and automatic conversion of language levels for UVL models. When embedding the library, developers can simply specify the supported language levels of their tool and the library automatically converts unsupported levels. We provide a video showing the usage of our tool.[1]

2 UNIVERSAL VARIABILITY LANGUAGE

The Universal Variability Language (UVL) is a community effort towards a unified textual format for variability models [2, 16]. Listing 1 shows a variability model describing a simplified PC in UVL notation. Each UVL model describes the set of features as a tree hierarchy. The core language supports the following parent-child relationships: *alternative*, *or*, *mandatory*, and *optional*. In our running example, `PC` is the root feature with three mandatory children `RAM`, `CPU`, and `Power Unit`, and one optional child `Designated GPU`. In addition to the tree hierarchy, additional constraints in propositional logic can be specified. The following operators are supported: `&` (and), `|` (or), `=>` (imply), and `<=>` (iff). In our running example, a `Designated GPU` requires the `Large` power unit to be selected.

Listing 1: UVL Example

```
features
    PC
        mandatory
            RAM
                or
                    "8GB"
                    "16GB"
            CPU
            "Power Unit"
                alternative
                    Large
                    Small
        optional
            "Designated GPU"

constraints
    "Designated GPU" => Large
```

Listing 2: UVL Imports

```
imports
    submodel as s

features
    PC
        mandatory
            s.RAM
            CPU
            "Power Unit"
                alternative
                    Large
                    Small
        optional
            "Designated GPU"

constraints
    "Designated GPU" => Large
```

UVL also comes with an import mechanism that can be used to reference UVL models from other files. For instance, the RAM subtree could be specified in a separate file. The submodel can then be attached in the feature tree as seen in Listing 2.

The core level of UVL is simple and still expressive enough for some popular feature-modeling tools, such as FeatureIDE [12]. Still, some use cases are difficult to express. For instance, in our running example, the need of selecting a large power unit probably depends on power usage and not specifically a designated GPU. Hence, the current constraint `"Designated GPU"` ⇒ `Large` is rather tedious to maintain. Adding more components with a high power usage requires updating the constraint. In the following, we present the extension mechanism of UVL that adds more sophisticated language properties while still keeping a sparse and simple base language.

3 LANGUAGE LEVELS IN UVL

Different use cases typically induce varying requirements for variability languages. Just adding additional language features to UVL, however, makes the integration into existing tools more complex.

[1] https://youtu.be/rah_z-uwWjU

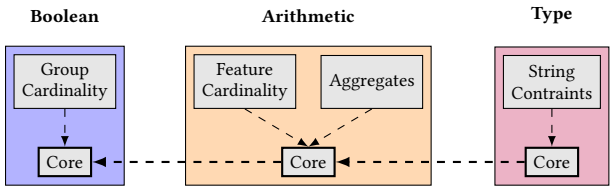

Figure 1: Language Levels in UVL

Each language feature would need to be handled by every tool integrating UVL. To support the inclusion only a subset of the language features, we propose a simple base language with optional extensions. To add more sophisticated language features, UVL employs optional language levels as suggested for variability languages in previous work [17].

Figure 1 shows the three language levels of UVL. The language levels group features by reasoning engines required to compute a solution. We expect that a variability-modeling tool that includes a SAT solver as backend can use all language features of the Boolean level but not for the arithmetic level. Each major level consists of a language core with some features and additional optional minor features. In the following, we present each of the three major language levels and their minor language levels.

3.1 Boolean Level

The basic UVL language presented in section 2 is the core of the *Boolean level*. As consequence, the core Boolean level consists of the language features described above: group keywords (e.g., *or*) and propositional cross-tree constraints. In addition, the Boolean level contains a sublevel for *group cardinality*. Instead of a group-type, the minimum and maximum number of features can be specified with `[min..max]`. For instance, the *or* group specifying the type of `RAM` modules could also be specified as `[1..2]`.

3.2 Arithmetic Level

The *arithmetic level* introduces numeric constraints to UVL. Expressions with the following operators are supported: `+`, `-`, `*`, `/`, `==`, `>`, and `>=`. These operators can be applied to create terms over constants and to numerical feature attributes as variables. Hereby, only terms resolving to Boolean are supported to be then used in the propositional constraints.

The minor level *feature cardinality* enables selecting one feature multiple times. Hereby, the same syntax as for group cardinalities is used (i.e., a feature can be selected `[min..max]` times). In our running example adapted for SMT (cf. Listing 3), between one and four 8GB RAM modules can be included.

The second arithmetic minor level *aggregate functions* enables the usage of the aggregate functions *sum* and *avg*. Both functions require the name of a numeric feature attribute as input. The aggregate functions then consider the attribute values with that name of all *selected* features. In addition to the attribute name, the functions can take a feature as second parameter. Then, only the feature and its descendants are considered for the aggregation. In Listing 3, an overall sum of required power larger than 120 requires the large power unit.

Listing 3: SMT Language Level in UVL

```
features
    PC
        mandatory
            RAM
                or
                    "8GB" cardinality [1..4] {Power 5}
                    "16GB" {Power 8}
            CPU {Power 100}
            "Power Unit"
                alternative
                    Large
                    Small
        optional
            "Designated GPU" {Power 50}

constraints
    sum(Power) > 120 => Large
```

Listing 4: Type Language Level in UVL

```
features
    PC
        mandatory
            RAM
                or
                    "8GB" {Power 5}
                    "16GB" {Power 8}
            Boolean CPU {Power 100, Manufacturer 'Intel'}
            "Power Unit"
                optional
                    String Manufacturer
                    Integer Watt
        optional
            "Designated GPU" {Power 50}

constraints
    sum(Power) < Watt
    CPU.Manufacturer == Manufacturer
```

3.3 Type Level

The *type level* introduces new feature domains, namely *real*, *integer*, and *string* features. In the previous levels, features are always boolean (i.e., can only be assigned true or false during configuration). For, e.g., an integer feature, a user may select an arbitrary integer as value. In the running example enriched with type level features (Listing 4), an integer value can now be configured for Watt. The aggregate functions (e.g., sum) can also be applied to numerical features in addition to attributes. Furthermore, the manufacturer can be configured with a string input. The domain of Boolean features can be explicitly set but is otherwise considered as default (e.g., CPU and RAM are both Boolean features).

The minor level *String constraints* introduces cross-tree constraints over string features and string constants. The following operators are supported: `len(<string>)` and == equality. The `len` function evaluates to a number and can be used in any arithmetic expression supported in the arithmetic level.

3.4 Explicit Declaration of Levels

In general, `UVLParser` automatically detects the used language levels. In some cases, it may make sense to actively chose supported language levels and explicitly declare them as shown in Listing 5. The *include*-keyword can be used to enable either just the core of

Listing 5: Include Mechanism in UVL

```
include
    Boolean
    Arithmetic.aggregate-functions
    Type.*
```

a major level (<major>), specific minor levels (<major>.<minor>), and all minor levels (<major>.*). Note that the respective core level for a minor level is always implicitly included. If modelers then introduce a language feature that is part of an excluded level, they get a warning. Otherwise, their change may not be supported by the employed reasoning engine. By default (i.e., no includes are specified), all language constructs are allowed.

4 CONVERSION STRATEGIES

With our notion of language levels, a major problem is still unsolved. Exchanging variability models with differing language levels is not possible without, possibly immense, overhead or information loss [8]. For instance, models with SMT-level language features cannot be directly used in a tool supporting the Boolean level.

To enable automated exchange, we thus employ conversion strategies between the language levels. The idea is to replace unsupported language features with syntactically different but semantically equivalent constructs. Using our parser, a tool can specify the supported language levels and each construct beyond those levels is converted. As lower language level constructs can always be used in higher language levels, we limit the conversion strategies to one direction for now. In some cases, the conversion of a language level may result in an exponential blowup [4]. As fallback, users can decide to drop the language constructs instead of converting them.

Naively implementing conversion strategies requires one conversion from each level to each other level. To reduce the number of required conversions but still be able to convert each level to lower ones, we apply a transitive approach. The dashed arrows in Figure 1 indicate a one-direction conversation between two language levels. Each major level requires a conversion to the next lower major level (e.g., *Arithmetic* to *Boolean*). Minor levels need a conversion to their respective core (e.g., *String Constraints* to *Type* core).

Table 1 gives a short overview on the different conversions employed. For instance, we convert group cardinalities by enumerating the valid feature combinations in propositional cross-tree constraints. Due to space restrictions, we refer to the repository for further insights on the implementation of conversion strategies.[2] With our conversion strategies, we translate language constructs according to their semantics following the literature. However, for feature cardinality, the interaction between a feature with a cardinality or its children and cross-tree constraints is not always clear [5, 14]. For UVL, we adopted the approach of creating repeated subtrees for the cardinalities and repeat cross-tree constraints with indexed features [5].

5 TOOL SUPPORT

Our tool `UVLParser` enriched with language levels and conversion strategies is available as Java library.[2] The library is realized

[2] https://github.com/Universal-Variability-Language/uvl-parser-java

Table 1: Conversion Strategies

	Level	Target Level	Conversion Strategy
Boolean	Core	-	-
	Group Card.	Boolean Core	Enumerating constraints
Arithmetic	Core	Boolean Core	Enumerating constraints
	Feat. Card.	Arithmetic Core	Repeated subtrees for feature instances
	Aggregates	Arithmetic Core	Expand (e.g., $a_1 + \ldots + a_n$ for sum)
Type	Core	Arithmetic Core	Non-Boolean as feature attributes
	String Const.	Type Core	Dropped

Listing 6: Library Usage in Java

```
// Read
UVLModelFactory uvlModelFactory = new UVLModelFactory();
FeatureModel featureModel = uvlModelFactory.parse(Files.readString("input.uvl"));

// Convert unsupported language levels
uvlModelFactory.convertExceptAcceptedLanguageLevels(featureModel,
        ↪ supportedLanguageLevels);

// Write
Files.write(Paths.get("output.uvl"), featureModel);
```

as Maven project. Thus, the library can be embedded into other projects based on Maven or exported as .jar. `UVLParser` is already integrated in FeatureIDE [12] and TRAVART [7]. The parsing is based on ANTLR [13]. Hence, a parser supporting the language levels could also be generated for other target languages, such as Python. Enabling the language levels and conversion strategies would, however, require additional effort.

Usage for Variability Modeling Tools. The API of `UVLParser` can be used to read and write UVL models and automatically apply conversion strategies. Internally, a UVL model is stored as feature model enriched with some semantic information to enable conversions and simplify external usage. Listing 6 showcases the usage of our library for (1) reading an UVL file, (2) converting unsupported language levels, and (3) writing the updated model to a new file. When integrating the parser library into a new tool, the developer just needs to specify their supported language levels to automatically convert UVL models.

6 RELATED WORK AND TOOLS

UVL Parsers. In previous work, multiple research groups, including authors of this work, implemented parsers based on Clojure [16] and Python [9]. Both parsers do not support any language levels or conversion strategies. For compatibility, `UVLParser` closely follows the respective grammars of the existing parsers for core of UVL. In recent work, we implemented a Rust-based language server protocol for UVL that also includes a parser.[3] The supported syntax is equivalent to the syntax of the parser presented here including the language levels. However, conversion strategies are not supported.

Variability Languages. Several other variability languages have been proposed and used in practice [1, 15]. ter Beek et al. [1] provide an overview on structural properties of available textual variability languages. While some other variability languages support the language features UVL has, none of them comes with a notion of language levels. Furthermore, none of the other languages supports conversion strategies.

[3] https://github.com/Universal-Variability-Language/uvl-lsp

7 CONCLUSION AND FUTURE WORK

The Universal Variability Language is a community effort towards a widely adopted textual format for specifying variability models. In this work, we present an extension mechanism based on two main components. Language levels facilitate a simple core language with optional, more complex extensions. Conversion strategies vastly simplify exchange between tools that supported different language levels. In the future, we plan to further improve the conversion strategies for better scalability in practice. Furthermore, we envision to maintain a tight cooperation with the community to discuss and realize other levels for UVL.

Acknowledgements. This work bases on the thesis of Vill [18]. Partially supported by FEDER/Ministry of Science and Innovation/Junta de Andalucía/State Research Agency with the following grants: *Data-pl* (PID2022-138486OB-I00), TASOVA PLUS research network (RED2022-134337-T), *METAMORFOSIS* (FEDER_US-1381375). The financial support by the CDG, the BMDW and the National Foundation for Research, Technology and Development is gratefully acknowledged.

REFERENCES

[1] Maurice H. ter Beek, Klaus Schmid, and Holger Eichelberger. 2019. Textual Variability Modeling Languages: An Overview and Considerations. In *MODEVAR*. ACM, 151–157.
[2] David Benavides, Rick Rabiser, Don Batory, and Mathieu Acher. 2019. First International Workshop on Languages for Modelling Variability (MODEVAR 2019). In *SPLC*. 323–323.
[3] Thorsten Berger and Philippe Collet. 2019. Usage Scenarios for a Common Feature Modeling Language. In *SPLC*. ACM, 174–181.
[4] Paul Maximilian Bittner, Thomas Thüm, and Ina Schaefer. 2019. SAT Encodings of the At-Most-k Constraint – A Case Study on Configuring University Courses. In *SEFM*, Peter Csaba Ölveczky and Gwen Salaün (Eds.). Springer, 127–144.
[5] Krzysztof Czarnecki and Chang Hwan Peter Kim. 2005. Cardinality-Based Feature Modeling and Constraints: A Progress Report. In *SF*. 16–20.
[6] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An Efficient SMT solver. In *TACAS*. Springer, 337–340.
[7] Kevin Feichtinger, Johann Stöbich, Dario Romano, and Rick Rabiser. 2021. TRAVART: An Approach for Transforming Variability Models. In *VaMoS*. ACM, Article 8, 10 pages.
[8] Kevin Feichtinger, Chico Sundermann, Thomas Thüm, and Rick Rabiser. 2022. It's Your Loss: Classifying Information Loss During Variability Model Roundtrip Transformations. In *SPLC*. ACM, 67–78.
[9] José A. Galindo and David Benavides. 2020. A Python Framework for the Automated Analysis of Feature Models: A First Step to Integrate Community Efforts. In *MODEVAR*. ACM, 52–55.
[10] 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. *CoRR* (2018).
[11] Daniel Le Berre and Anne Parrain. 2010. The Sat4j Library, Release 2.2. *JSAT* 7, 2-3 (2010), 59–64.
[12] Jens Meinicke, Thomas Thüm, Reimar Schröter, Fabian Benduhn, Thomas Leich, and Gunter Saake. 2017. *Mastering Software Variability with FeatureIDE*. Springer.
[13] Terence J. Parr and Russell W. Quong. 1995. ANTLR: A Predicated-LL(k) Parser Generator. *SPE* 25, 7 (1995), 789–810.
[14] Clément Quinton, Daniel Romero, and Laurence Duchien. 2013. Cardinality-Based Feature Models with Constraints: A Pragmatic Approach. In *SPLC (SPLC '13)*. ACM, 162–166.
[15] Mikko Raatikainen, Juha Tiihonen, and Tomi Männistö. 2019. Software Product Lines and Variability Modeling: A Tertiary Study. *JSS* 149 (2019), 485–510.
[16] 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 *SPLC*. ACM, 136–147.
[17] Thomas Thüm, Christoph Seidl, and Ina Schaefer. 2019. On Language Levels for Feature Modeling Notations. In *MODEVAR*. ACM, 158–161.
[18] Stefan Vill. 2022. *Language Levels for the Universal Variability Language: An Extension Mechanism and Conversion Strategies*. Bachelor's Thesis. University of Ulm.