The High-Level Variability Language: An Ontological Approach

Angela Villota, Raúl Mazo, Camille Salinesi

The High-Level Variability Language: An Ontological Approach

Angela Villota
CRI, Université Panthéon-Sorbonne
Paris, France
i2t, Universidad Icesi
Cali, Colombia
apvillota@icesi.edu.co

Raúl Mazo
CRI, Université Panthéon-Sorbonne
Lab-STICC, ENSTA Bretagne
France
GIDITIC, Universidad EAFIT
Medellín, Colombia
raul.mazo@univ-paris1.fr

Camille Salinesi
CRI, Université Panthéon-Sorbonne
Paris, France
camille.salinesi@univ-paris1.fr

ABSTRACT

Given its relevance, there is an extensive body of research for modeling variability in diverse domains. Regretfully, the community still faces issues and challenges to port or share variability models among tools and methodological approaches. There are researchers, for instance, implementing the same algorithms and analyses again because they use a specific modeling language and cannot use some existing tool. This paper introduces the High-Level Variability Language (HLVL), an expressive and extensible textual language that can be used as a modeling and an intermediate language for variability. HLVL was designed following an ontological approach, i.e., by defining their elements considering the meaning of the concepts existing on different variability languages. Our proposal not only provides a unified language based on a comprehensive analysis of the existing ones but also sets foundations to build tools that support different notations and their combination.

CCS CONCEPTS

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

KEYWORDS

domain specific language, variability language, variability specification

ACM Reference Format:
Angela Villota, Raúl Mazo, and Camille Salinesi. 2019. The High-Level Variability Language: An Ontological Approach. In *23rd International Systems and Software Product Line Conference - Volume B (SPLC '19), September 9–13, 2019, Paris, France.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3307630.3342401

1 INTRODUCTION

Variability modeling is an extensively studied subject. Research in this subject includes several variability modeling languages that have been proposed in academia and industry [3]. Most research in the area focuses on feature-based modeling languages since the introduction of FODA [13]. However, other modeling approaches exist, variation point-based models [20], decision-based models [6], goal-oriented models [19], constraint-based languages [24] and industrial languages (e.g., Kconfig[28], Gears[15]) can be used to describe variability. These proposals have contributed to a universe of languages, notations, transformations, and tools supporting the creation of variability models.

Variability modeling languages are neither completely different nor completely the same. Indeed, these languages share most variability concepts such as variability units, and constraints, but also differ in concepts that are relevant to particular domains or modeling styles. For example, Figure 1 presents three variability models written in different languages: *FODA* [13], *Dopler* [6], *OVM* [20]. These models have (1) different structures (e.g., hierarchical, non-hierarchical); (2) different types of variability units (e.g., Boolean, non-Boolean); (3) heterogeneous rules (e.g., cross-tree constraints, visibility, and validity conditions); among others.

Currently, variability modeling relies upon existing domain-specific languages and modeling tools. These tools are developed and taught in-house and frequently are used only by the few people associated with the development team. This diversity of languages and tools causes lack of portability in models and interoperability issues between SPL engineering tools. One of the poor consequences is that modeling tools require numerous parsers and transformations that might cause expressiveness loss. To solve this gap, the Common Variability Language (CVL) was proposed as a standard language for variability modeling [12]. However, this initiative did not succeed, and the community still faces issues caused by the diversity of languages, dialects, and tools.

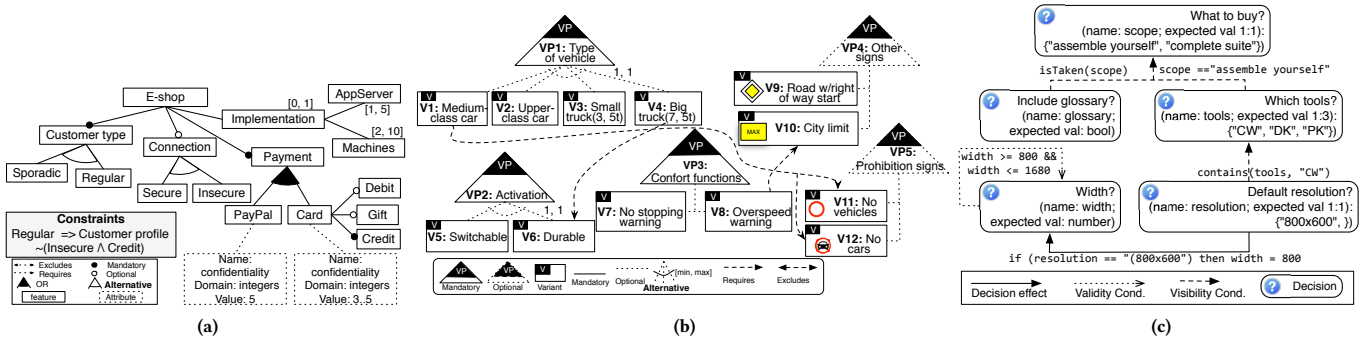
This paper presents our proposal for moving forward the initiative of defining a standard variability language. Particularly, this paper introduces: (i) a glossary of basic concepts from variability modeling languages, and (ii) a variability language able to describe these concepts and to work with/for/in combination with most languages and tools, the *High-Level Variability Language (HLVL).*

Developing a standard language can be an effort-intensive endeavor. Therefore, we followed an ontological approach for conceptualizing and structuring knowledge about variability modeling concepts. In this case, an ontological approach is favorable to determine a set of constructs to describe variability comprehensively and to define the characteristics of a language capable of representing constructs from different variability languages.

HLVL belongs to an ongoing project in which we envision a language capable of supporting concepts of many variability languages to reduce interoperability and sharing issues. The general idea is that variability models can be compiled into an intermediate

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

ACM ISBN 978-1-4503-6668-7/19/09…$15.00
https://doi.org/10.1145/3307630.3342401



(a) (b) (c)

Figure 1: Variability models using three graphical languages. (a) FODA [13] with the extension proposed in [2]. (b) Orthogonal Variability Model (OVM) [20]. (c) Dopler Modeling Language (DoplerML) [6]

form and can be interpreted on other tools. In fact, if variability models are written in an intermediate language, such as HLVL, they could be integrated analyzed and configured into a single model in an integrated way. This research gives continuity to previous works [8, 17, 19, 24] that exploit the idea of relying on a variability language that unifies existing notations to provide genericity to the methods, techniques, and tools used for modeling, analysis, and configuration.

Section 2 the ontological approach used to design HLVL. Section 3 presents our proposal for a glossary of variability concepts. Section 4 introduces the syntax of HLVL and shows how HLVL supports different styles of variability modeling using examples. Sections 5 and 6 present the discussion and related work, respectively, and Section 7 concludes the paper concludes the paper with our final remarks.

2 DESIGNING HLVL FOLLOWING AN ONTOLOGICAL APPROACH

The ontological approach followed to structure the domain knowledge for designing the HLVL consisted of three steps depicted in Figure 2 and described below. Note that though this proposal unifies concepts from different variability languages, it does not subsume all variability languages, nor is our language capable of representing every single variability language. A broader ontological comparison would be necessary to produce such a language; which is a cumbersome and not scalable task.

Ontological analysis: in this step, we conducted an ontological analysis of the expressiveness of an initial version of the variability language. In this study, we used the variability patterns introduced by Asadi et al. [1] to determine the criteria for completeness and clarity from the ontological expressiveness perspective. The results showed that (1) the language closely represents the concepts in the ontological framework. However, some variability concepts should be integrated for obtaining a 100% level of completeness. (2) The language's high-level of abstraction impacts its clarity because several elements in the ontology are represented by the same language construct. A broader description of this analysis and its results are available in [27].

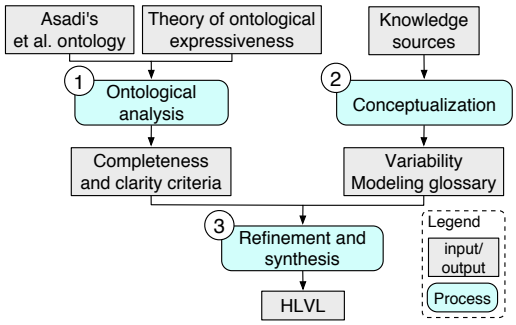


Figure 2: Process for designing HLVL

Conceptualization: this step consisted of the identification and review of research about variability modeling in literature. The studies selected for this step were gathered while conducting a systematic mapping study, an upcoming publication from the authors[1]. More particularly, the conceptualization included languages that (1) have been transformed to logic or constraint programming to automate analysis tasks; and (2) their semantics is formally defined, and were included in other conceptualization studies. Hence, this step included feature-oriented languages [5, 13, 14, 26]; variation point oriented languages [8, 12, 20, 22, 23]; and decision-oriented languages [4, 6, 7, 25]. Finally, the review included proposals introducing constructs for modeling complex variability relations, such as conditional and quantified constraints [8, 14, 21]. Based on the reviewed literature, a collection of variability modeling concepts were organized and structured in a glossary. Section 3 presents the result obtained in this step.

Refinement and synthesis: using the results from the previous steps, we proposed the characteristics of HLVL language. Section 4 presents the HLVL in detail.

3 VARIABILITY MODELING GLOSSARY

Variability languages enable the modeler to answer two questions about the product line to be modeled: *what does vary?* and *how does it vary?* [20]. These languages provide a collection of *constructs*

[1]Protocol available at http://bit.ly/2IVBde5

enabling the modeler (1) to identify and document the variable items in a product line; (2) to identify the set of possible options or variants associated to variable items in the system; (3) to identify the rules for determining how items can be combined into new configurations; and finally (4) to produce variability models. Language constructs in variability languages define the *variability units* and the *variability relations*.

3.1 Variability Units

Variability Units (VUs) are the key concept used to model variability in a language [4]. VUs represent variable items in a system or domain, that is, those aspects that must be chosen by the customer or engineer in a configuration process. For example, *features* are the VUs in feature models, *decisions* in decision models, and *variation points*, *variants* are the units in OVM models. VUs are characterized by their type and multiplicity.

Type. The type of a VU is defined by the number of variants it represents. VUs are Boolean when they are associated with exactly two options: *{selected, unselected}*, as in feature models. Non-Boolean VUs have more than two options, as *attributes* in attributed-based feature models, or *decisions* in decision models. Some languages such as Gears allow the declaration of complex data-types such as enumerations, sets, and records [15].

Multiplicity. It represents the number of instances of a VU that may appear in a configuration. Then, VUs are annotated using cardinalities in a UML style. For example, features with the interval $[m, n]$ have at least m and at most n instances in a configuration [5].

3.2 Variability relations

Variability relations determine the rules to select and recombine items into new products. Variability relations are often presented as dependencies or constraints and are usually denoted graphically (i.e., using arrows) or textually (i.e., logic formulas, OCL). Variability relations can be classified as follows:

Inclusion/exclusion rules. Are the set of basic rules for describing variability in a product line. In this set, we placed the rules for defining conditional inclusion/exclusion and commonalities.

Conditional inclusion/exclusion are rules for restricting the inclusion/exclusion of variable items given a condition. This condition can be simple as in FODA *requires* and *excludes* constructs where the inclusion/exclusion of a feature B is conditioned to the inclusion of a feature A. Languages such as CO-OVM[8], and extended-feature models in [14] allow the usage of complex expressions to condition the inclusion/exclusion of variable items using logical expressions.

Commonality. Rules for defining items that always appear in any configuration. This rule is implicit in some languages (e.g., root feature in feature models, mandatory variation points in OVM) or inexistent (e.g., decision models). This is different from calculating the set of core items, an analysis operation that requires extra processing.

Hierarchy - Decomposition. In decomposition relations, one item plays the role of parent, and the other is the child. This parent-child relation imposes a constraint in the configuration because no child can be part in a configuration without the inclusion of its parent. There are two types of decompositions:

On-to-one decompositions relate pairs of items. There are two types of decompositions: *mandatory* and *optional*. In mandatory decompositions, the child is included in all products in which its parent appears. Instead, in optional decompositions, the child can be optionally included in all products in which its parent appears.

One-to-many decompositions relate one parent and a group of children. This relation restricts the minimum and the maximum number of children that may be included in a configuration when their parent is selected.

Hierarchy - visibility. Visibility rules condition the availability of other variability items. Visibility relations are considered a type of hierarchical relations because they are used to compartmentalize items, as in different views, e.g., for different stakeholders [4]. Visibility relations are a common construct in decision-based languages such as Dopler [6].

Constraint expressions. Constraint expressions are used to include complex rules between variable items in a product line model. This rules can be composed using relational, arithmetic and global operators among others. These constraint expressions are often used to specify extra-functional information or to include contextual rules. For instance, the extended-feature models in [14] allow the inclusion of expressions and operators for adding constraints between feature's attributes and instances.

4 THE HIGH-LEVEL VARIABILITY LANGUAGE

HLVL is a variability modeling language that addresses the following three requirements:

Rq1. HLVL provides constructs to model the concepts in the variability modeling glossary to describe variability comprehensively.

Rq2. HLVL specifies variability models in different approaches such as feature-oriented, variation point oriented, and decision-oriented modeling.

Rq3. HLVL's syntax should be understandable for humans to create and edit models, but also should be formally defined to be generated and interpreted by modeling tools.

The following subsections present an overview of the HLVL syntax (see Section 4.1) and further examples of the HLVL's usage (see Section 4.2).

4.1 Syntax

Variability models in HLVL are defined in terms of constructs called **elements**, **variants**, and **variability relations**. These constructs map the concepts defined in the variability modeling glossary previously presented in Section 3. The collection of elements, variants, and variability relations describing a model in HLVL conform a **script**. Scripts in HLVL start with the keyword `model` followed by an identifier. Each block of the script also starts with a keyword (i.e., `elements`, and `relations`). A simple E-shop product line adapted from [22] (see Figure 1a) serves us to illustrate the HLVL's syntax (see Table 1). This example describes a basic scenario that will grow as we introduce language constructs. These constructs are formally defined using the BNF grammar summarized in Table 2.

Table 1: Running Example

The online-shopping product line is a collection of similar e-commerce web applications. All products in the E-shop domain must have a module to handle the customer type, a module for manage the payment, and optionally, some modules for managing users' connection. In this product line, the payment method may be provided by PayPal services, or card. Additionally, the system must guarantee a secure connection when the transaction is performed by a regular customer o when the payment is performed using a credit card. Finally, each payment module has a confidentiality level that represents the privacy level of payment details. The confidentiality levels range from one to five.

4.1.1 Elements and Variants. Elements are the variability unit in HLVL. Elements in HLVL are typed, and optionally include a keyword to define attributes or comments introduced by the modeler. HLVL supports **boolean**, **integer**, and **symbolic** data types. Each element is associated with a set of variants that represents the available choices. The configuration process selects exactly one choice from the set of variants (i.e., enumeration). The variants associated to an element are declared using intervals or lists of values regarding the data type. The following example shows the declaration of a group of Boolean elements and a symbolic element. As shown in the example, in HLVL, Boolean variants do not require an explicit declaration (syntactic sugar).

```
model eShop
elements:
  boolean connectionType, secureConnection, insecureConnection,
      payment, payPal
  symbolic customerType variants: ['sporadic', 'regular']
      comment: {"This element represents the customer type"}
```

Note that HLVL's identifiers can be composed in a flexible programming language fashion. Then, the only rules regarding identifiers are that they cannot start with digits, special characters, or contain spaces.

Attributes. Elements can be used to represent attributes. In HLVL, we differentiate an attribute from a regular element using the keyword **att** in the element declaration. For example, to define an integer attribute representing confidentiality in HLVL, we write the following:

```
att integer confidentiality variants: 1..5
att integer confBounded is 2
```

The example also shows the definition of `confBounded` as a bounded attribute to a value by the keyword **is**. We included this definition to support the simplification of attributes introduced by some tools.

Multiplicity. Elements in HLVL can have multiple instances with local semantics as described in [18]. Syntactically, multiplicities are declared as properties for dependency relations as we explain below.

4.1.2 Variability Relations. Variability relations in HLVL can be used for (1) defining inclusion/exclusion rules; (2) describing hierarchies; (3) constraining the visibility of other variability relations; and (4) including complex expressions such as arithmetic, logic, and relational. Table 2 presents the syntactic rules for HLVL's variability relations (i.e., $R7 - R17$). As shown in this table, all variability relations in HLVL contain an identifier for referencing.

Table 2: Syntax for variability relations in HLVL

⟨*element*⟩ ::=	{**att**}? ⟨*data_type*⟩ E_1 **variants:** ⟨*variants*⟩ {**comment:***String*}? |	($R1$)
	{**att**}⟨*data_type*⟩ E_1 **is** ⟨*value*⟩	($R2$)
⟨*variants*⟩ ::=	⟨*integer*⟩..⟨*integer*⟩ |	($R3$)
	[⟨*value*⟩{,⟨*value*⟩}*]	($R4$)
⟨*data_type*⟩ ::=	**boolean** | **integer** | **symbolic**	($R5$)
⟨*value*⟩ ::=	**true** | **false** | {0..9}+ | '⟨*string*⟩'	($R6$)
⟨*sentence*⟩ ::=	⟨*identifier*⟩ : ⟨*relation*⟩	($R7$)
⟨*relation*⟩ ::=	**common**($E_1, E_2, \ldots, E_k$) |	($R8$)
	mutex(E_1, E_2) |	($R9$)
	mutex(⟨*expression*⟩,$E_1, \ldots, E_k$) |	($R10$)
	implies(E_1, E_2) |	($R11$)
	implies(⟨*expression*⟩,$E_1, \ldots, E_k$) |	($R12$)
	decomposition(P,[$C_1, C_2, \ldots, C_k$], [m,n]) |	($R14$)
	group(P,[$C_1, C_2, \ldots, C_k$], [m,n]) |	($R15$)
	visibility(⟨*expression*⟩,[$R_1, \ldots, R_k$])	($R16$)
	expression(⟨*expression*⟩)	($R17$)

Commonality. HLVL provides a construct to declare common elements in a product line explicitly. In the running example, the modules for handling the customer type and the payment are always part of an E-shop. In HLVL, this is expressed as follows:

```
com1: common(customerType, payment)
```

Inclusion/Exclusion relations. HLVL provides different constructs to describe inclusion and exclusion rules in particular, *constraint expressions*. Constraint expressions in HLVL are useful for including complex rules between elements in the variability model using logic, relational, arithmetic, and global operators. These complex rules are written in the HLVL's expressions language (cf. Table 3), that is also used to write the conditions in other variability relations. For example, to restrict the confidentiality levels of the payment by card module to be between 3 and 5, we write the following relation in HLVL:

```
exp1: expression(3 <= card.confidentiality AND
                 card.confidentiality <= 5)
```

Conditional exclusion/exclusion relations can optionally be described using language constructs. To this purpose, HLVL provides the keywords **mutex** and **implies**. Through language constructs, HLVL supports two types of conditional exclusion: *mutual exclusion* and *guarded exclusion*. Consider the following example:

```
m1: mutex(creditCard, insecureConnection)
m2: mutex(customerType='sporadic',[ giftCard , creditCard ])
```

Here, $m1$ represents the mutual exclusion of the credit card payment and insecure connection. Then, these two elements cannot be part of the same configuration. Also, the guarded exclusion $m2$ defines a condition to exclude the payment by gift card and debit card for sporadic customers. Guarded exclusion may have complex conditions using HLVL's expressions language. Then, whenever the condition is satisfied the group of elements won't be included in a configuration.

Similarly, HLVL supports *implication* and *guarded implication* using constructs as follows.

Table 3: Syntax of the expressions language

⟨*expression*⟩ ::=	~ ⟨*boolExp*⟩ | ⟨*boolExp*⟩ | ⟨*assignExp*⟩
⟨*boolExp*⟩ ::=	⟨*boolVal*⟩ | ⟨*boolExp*⟩ ⟨*logicOp*⟩ ⟨*boolExp*⟩ | ⟨*relational*⟩
⟨*relational*⟩ ::=	⟨*arithmetic*⟩ ⟨*relationalOp*⟩ ⟨*arithmetic*⟩
⟨*arithmetic*⟩ ::=	⟨*numericVal*⟩ | ⟨*numericVal*⟩ ⟨*arithmeticOp*⟩ ⟨*numericVal*⟩
⟨*boolVal*⟩ ::=	⟨*name*⟩ | `true` | `false`
⟨*numericVal*⟩ ::=	⟨*name*⟩ | ⟨*integer*⟩
⟨*logicOp*⟩ ::=	`AND` | `OR` | `=>` | `<=>`
⟨*RelationalOp*⟩ ::=	`=` | `!=` | `>` | `>=` | `<` | `<=`
⟨*arithmeticOp*⟩ ::=	`+` | `-` | `*` | `/`
⟨*name*⟩ ::=	⟨*name*⟩.⟨*name*⟩ | ⟨*identifier*⟩

imp1:implies(payPal, secureConnection)
imp2: implies(customerType='regular', [secure, customerProfile])

In this example, we use *imp*1 to represent that the inclusion of PayPal payment implies the use of the secure connection (i.e., requires). Also, *imp*2 represents the inclusion of the modules for handling secure connection and customer's profile, conditioned to the selection of a regular customer. Conditions in guarded inclusions are written using constraint expressions.

Hierarchy-Decomposition. Although HLVL is not a language where hierarchical relations are essential for composing models, it offers a set of constructs to describe one-to-one (parent-child), and one-to-many (parent-children) decompositions.

One-to-one decompositions in HLVL contain the keyword `decomposition` followed by the names of the parent and the child element together with a cardinality $[m, n]$. This cardinality is a *multiplicity annotation* and is used to bound the number of instances of the child element. Decompositions of type mandatory and optional can be considered special cases with the cardinalities $[1, 1]$ and $[0, 1]$, respectively. In the running example, the relations stating that the gift-card and debit-card modules are optional and the credit-card module is mandatory are written in HLVL as follows:

dc1: decomposition(card, [giftCard, debitCard],[0,1])
dc2: decomposition(card, [creditCard],[1,1])

To illustrate dependencies with cardinality $[m, n]$, let us now make an addition to the running example: Suppose that the E-shop product line is implemented using between one and five application servers (e.g., Glassfish, Tomcat, Jetty, etc) supported by minimum two and maximum ten machines (see Figure 1a). In HLVL, this is written as follows:

dc3: decomposition(implementation, [appServer], [1,5])
dc4: decomposition(implementation, [machines], [2,10])

Decompositions with cardinality $[0, 1]$ are used to associate Boolean elements to one or more attributes. Let's extend the example, including an attribute for the type of security certificate in the payment modules. The association of elements to attributes in HLVL is written as follows.

a1:decomposition(payPal,[confidentiality , certificateType],[1,1])
a2:decomposition(card,[confidentiality , certificateType],[1,1])

The inclusion of these relations enable the qualified names `payPal.confidentiality`, `payPal.certificateType`, `card.confidentiality`, and `card.certificateType` to differenciate each attribute. Also, note that elements cannot represent attributes and have multiplicities at the same time.

one-to-many decompositions contain the `group` construct, the identifier of the parent, the children identifiers enclosed in brackets followed by an interval representing the cardinality. This cardinality is used to specify the minimum and the maximum number of children that can appear in a product. For example, in the E-shop product line, the variability in the payment method can be modeled using a group relation as follows:

g1: group(payment, [payPal, card], [1,*])

In this example, the cardinality `[1,*]` denotes that at least one, and at most the number of children can be selected.

Visibility. Visibility relations in HLVL are rules to condition the availability (i.e., hide) of a group of elements and their relations with similar semantics than visibility rules in decision models [7]. These relations are declared starting with the keyword `visibility` followed by a constraint expression and the identifiers of the elements this condition hides. For example, let's imagine that the implementation characteristics of the E-shop are associated with the company business (i.e., service seller or product seller). Then, elements `implementation`, `appServer`, and `machines` will be visible only if the company commercializes services. We can represent this in HLVL as follows:

v1: visibility (productType = 'services', [implementation, appServer, machines])

4.2 Other Examples in HLVL

The following subsections show how specific constructs of other notations are represented using HLVL. First, the excerpt of the model of the Radio Frequency Warner system (RFW) product line taken from [23] and written in OVM (Figure 1b). Second, the excerpt of the dopler model describing the variability of the Dopler tool suit taken from [16] (Figure 1c). The examples at their full extent are available at https://github.com/angievig/Coffee/tree/master/HLVL/Examples/MODEVAR.

4.2.1 Modeling Variation Point Languages in HLVL. Variation Points (VP) and variants can be modeled using Boolean elements in HLVL. For example, the variation point VP5 in Figure 1b representing the prohibition signs and its variants, the no vehicles sign (V11), and no cars sign (V12) can be modeled in HLVL as follows:

boolean VP5 comment:{"Prohibition signs"}
boolean V11 comment:{"No vehicles"}
boolean V12 comment:{"No cars"}

In this example, we used the VP's identifier to name the element in HLVL and the keyword `comment` to include the extra information in the diagram.

Mandatory VPs, that is variation points that must always be bounded are declared using the `common` construct. In the RFW, `VP1`, `VP2`, and `VP3` are mandatory VPs, this is expressed in HLVL as follows:

c1: common(VP1, VP2, VP3)

The links between VPs and variants (i.e., mandatory, optional, and alternative) are one-to-one and one-to-many decompositions.

In HLVL, mandatory and optional links are represented using the **decomposition** construct. Alternative $[m, n]$ links are represented using the **group** construct. For instance, the optional relation between VP5 and V11, V12, and the alternative relation between VP1 and V1, V2, V3, V4 is written in HLVL as follows:

```
d1: decomposition(VP5, [V11, V12], [0,1])
d2: group(VP1, [V1, V2, V3, V4], [1,1])
```

Alternatively, VPs and variants linked by alternative relations with cardinality [1..1] can be represented in HLVL using a symbolic element for the VP and symbolic values for the variants in the group. For example, VP2 and V5, V6 can be written in HLVL as follows:

```
symbolic VP2 variants: ['V5', 'V6'] comment:{"Activation"}
```

Constraints in OVM models can me represented using the **implies** and **mutex** constructs. In the example, the implication (requires) between V8 and V10 is expressed as follows:

```
imp2: implies(V8, V10)
```

When the modeler choose to represent alternative [1..1] relations using symbolic elements, the constraints can be represented with constraint expressions and guarded implications in HLVL. Let's imagine that VP1 and VP2 are symbolic symbols, then the implications between pairs (V4, V6), (V1, V11), and (V1, V12) can be written in HLVL as follows:

```
exp1: expression(VP1 ='big truck' => VP2 ='durable')
imp1: implies(VP1 ='medium−class car', [V11,V12])
```

Attributes and complex constraints in the CO-OVM style [8] are represented in HLVL using constraint expressions and conditional implications.

4.2.2 Modeling Decision Models in HLVL. Decisions with cardinality 1 : 1 can be modeled using elements, their data types, and comments. To illustrate this, the scope and glossary decisions in the example are represented in HLVL as follows:

```
symbolic scope variants: ['assemble yourself', 'complete suite']
        comment: {"What to buy?"}
boolean glossary comment:{"Include glossary?"}
```

Additionally, decisions with cardinality 1 : N are represented using Boolean elements and a **group** relation. For example, the decision tools, its three variants, and its cardinality are written as follows in HLVL:

```
boolean tools, confWizard, decisionKing, projectKing
g1:group(tools ,[confWizard,decisionKing, projectKing ], [1,3])
```

Visibility conditions in decision models are modeled in HLVL using the **visibility** construct. In the example, the decision about the resolution is visible if the user decides to include the configuration wizard tool. Also, the glossary will be included after a decision about the scope is taken. These visibility conditions are expressed in HLVL as follows:

```
vis1: visibility (confWizard=true, [ resolution ])
vis2: visibility ( entailed (scope), [glossary ])
```

The function **entailed** is used to determine if the value of an element is already decided.

Decision effects in dopler models describe dependencies between decisions as rules triggering values for other decisions. For example, the rule determining that the selection of the resolution triggers the value of the width is written using constraint expressions in HLVL as follows:

```
e1: expression (( resolution ='800x600') => width = 800))
```

Validity conditions are the rules restricting the range of the values which can be assigned to a decision. In HLVL, these rules are written using constraint expressions. For example, the validity condition restricting the width as a number between [800, 1680] is written in HLVL as follows: the

```
val1: expression (width >= 800 AND width <= 1680)
```

5 DISCUSSION

5.1 Intermediate Language for Variability

One of the main characteristics of HLVL is that it contains constructs for comprehensively modeling variability concerning the concepts in the variability glossary presented in Section 3. Hence, HLVL can be used as used (1) as a specification language to create variability models; or (2) as an intermediate representation of models specified in other variability languages. Here, we borrow the concept of intermediate language from the compilers' domain. In this domain, intermediate languages are used to produce intermediate representations during the process of translating a source program into target code. Many compilers generate an explicit low-level or machine-like intermediate representation, which can be thought of as a program for an abstract machine, as in the case of the Java language.

The usage of an intermediate language for variability is a viable alternative to the interoperability problem because written in such a language, variability models can be easily shared or distributed. Then, modeling tools should be able to export and import models in the intermediate language, so modelers do not have to learn a new variability language, then, modeling tools can be used as they are today. An intermediate language for variability can be to variability modeling tools as the BibTeX format is for reference tools. That is, these managing references applications (e.g., Mendeley, Zotero, etc.) have their own formats and styles for managing references. Yet, these applications are also capable of importing and exporting BibTeX formats. Even electronic databases, for example, ACM data library, IEEE Xplore, Springer, Science Direct, citeSeer, etc. have their own way to store and display references. However, these electronic databases provide an option for downloading or exporting references in the BibTeX format. Moreover, if the rare cases that a publication has not an available a BibTeX format, it is possible to define its BibTeX because the language' syntax is simple. Also, there exits examples and documentation for the BibTeX notations are publicly available.

5.2 Many Languages, One Representation

To ensure the flexibility of the language, HLVL has syntactical elements for modeling Boolean and non-Boolean variability supporting the description of simple and complex variability models. Besides, HLVL supports different styles of variability modeling. As shown in the examples above, HLVL can be used to specify FODA models, attribute-based feature models, cardinality-based feature models, variation-point oriented models or even decision-based

models. In the conducted literature review, we observed that variability models are often enriched with ad-hoc constraints to gain expressiveness. These approaches contribute to the proliferation of new dialects and language extensions. Considering that the transformation of the base language into an HLVL model is viable, the new constraints can be introduced in HLVL. Then, HLVL can be a standard language to add variability relations not supported by current notations to enhance variability models without increasing the variability of variability languages.

The support of Boolean and non-Boolean variable items, the capability of model different variability languages, and the potential capability to enhance variability models let us envision HLVL as a viable language for integrating variability models described in different languages. Either written in the same editor in HLVL or created in different modeling tools, models from different sources can be integrated to be analyzed or configured.

5.3 What is Next for HLVL?

This paper presented the syntax, semantics, and usage scenarios of HLVL. However, the full design of the language comprises the formal definition of semantics and other syntactic aspects such as well-formedness rules and lexical syntax. To this purpose, we will consider the guidelines for defining modeling languages proposed by Harel and Rumpe in [11]. Also, we will examine the formal semantics for the variability languages supported by HLVL [7, 18, 26]. We will study carefully the semantics of visibility and multiplicity relations, as well as the effects in the configuration semantics produced by these relations.

Then, the next step in our research consists in the evaluation of the language focusing on expressiveness. This evaluation will measure the language's expressiveness from two different points of view. First, we will conduct a new ontological analysis to verify that this new proposal solves the issues reported in our previous work [27]. We are aware that the fact inclusion/exclusion relations can be represented by more than one language construct produces construct redundancy, which is one of the defects in conceptual modeling languages. However, we consider that this defect is preferable to the ambiguities created by having one construct mapping many variability concepts. These ambiguities may interfere in the transformation process to obtain the HLVL representation of a model initially written in another language.

In the second part of the evaluation, we will show that HLVL's constructs map the constructs of variability languages implemented by particular modeling tools. Section 4 presented three examples used to illustrate how concepts from different academic variability modeling languages can be represented using HLVL. In this evaluation, we are interested in providing transformation rules and algorithms to produce HLVL models using as input the formats produced by tools implementing those academic languages. The first step in this direction was the implementation of a concept-proof including an editor for HLVL models and the Java tools for translating variability models specified in two different tools (Code available at https://github.com/angievig/CoffeeProofOfConcept). This implementation was used to demonstrate the feasibility of the usage of HLVL as a modeling and intermediate language. We plan to apply this proposal in the reengineering of the VariaMos SPL tool suite.

6 RELATED WORK

In previous works [8, 17, 19, 24] we have developed the idea to provide full *genericity* to methods, techniques, and tools for variability modeling, analysis, and configuration. First, Salinesi et al. [24] proposed to use a constraint programming language to represent variability. This proposal relied on the expression power of constraint programming. However, the benefits of this language do not compensate the drawbacks in the design given the language's lack of usability and readability. At some point, to use this language resembled replacing a programming language by assembly language: regardless its benefits, to work with large scale assembly programs without a higher level, more abstract language is an unfeasible task.

More recent works [17, 19] introduced different levels of metamodels to provide a high-level view of the constraint language that serves as generic language. This proposal is fully implemented in the current VariaMos tool suite supporting different variability languages and providing tools for extensions. However, this is a complex approach with a lack of usability and poor tool performance. Dumitrescu et al. [8] developed a variant of SysML to address the design of cyber-physical systems considering industry standards. The variability language proposed in this work contains various constructs that are not relevant to variability specification. This paper, presents an ontological approach introducing HLVL, an agnostic variability representation that serves as modeling and intermediate language. We followed a compiler's approach where the language provides a formal syntax. This approach eases the generation of HLVL code from other tools.

Eichelberger and Schmid report a certain trend in product line engineering towards textual variability modeling languages [9]. The survey and analysis of textual variability languages presented in their work characterizes eleven textual languages, including their own proposal the INDENICA Variability Modeling Language (IVML). Among the results, the authors report that most textual variability languages support feature-oriented modeling and less frequently, variation point oriented modeling. In contrast, our proposal supports feature-oriented modeling, variation-point oriented modeling, and decision-based modeling. In consequence, the HLVL allows the creation of models in any of these styles of modeling. Moreover, to combine in a model constructs from different approaches that were traditionally exclusive to a set of modeling languages.

Galindo et al. [10] present an approach to ease the integration of variability models specified using different modeling styles, variability languages, and tools to perform configuration. They introduce the Invar approach to provide the user with a configuration tool that hides the different models, their semantics, and internal representation. Then, the configuration is performed by different tools and is orchestrated by an API that manages the communication between the configurator and the analysis and configuration tools. Alternatively, our proposal considers the integration of variability models using HLVL as an intermediate language to perform analysis and configuration operations.

7 CONCLUDING REMARKS

Migrating or integrating models built with different languages is challenging because many concepts and forms are not consistent among them. To define a unified language, we have been applying an ontological approach, i.e., we have analyzed feature-oriented, variation point oriented, and decision-oriented languages to define a glossary of concepts and propose a unified language based on this glossary. This paper introduces the High-Level Variability language (HLVL), a unified variability language defined following our ontological approach. Here, we presented the HLVL using an example containing complex rules considering Boolean and non-Boolean elements, attributes, multiplicities, and constraint expressions. Also, we show how HLVL supports different styles of variability modeling using two examples in different languages.

HLVL is a declarative language with a formally defined syntax that resembles programming languages. The formal definition of HLVL's syntax eases the code generation from other variability languages. Also, being a programming-like language, HLVL is a more human-readable language than other markup languages. However, we consider that the concrete syntax presented in this paper may evolve as a consequence of further validation and evaluation. We believe that the ontological approach in this research and the resulting unified language are viable alternatives to the interoperability issues evidenced by the product line community. This research contributes to reducing the lack of consensus in the concepts that should be included in variability languages. Also, it contributes to the proposal of a standard format useful for the portability of variability models.

Further discussion is required in this subject since the discussion was oriented from the modeling perspective, more particularly from an expressiveness perspective. However, the discussion should also focus on the characteristics of the variability language that ease the analysis and extraction of information from variability models. This other perspective and further evaluation of the HLVL are part of our ongoing project about the application of constraints for variability modeling and reasoning.

REFERENCES

[1] Mohsen Asadi, Dragan Gasevic, Yair Wand, and Marek Hatala. 2012. Deriving Variability Patterns in Software Product Lines by Ontological Considerations. In *Conceptual Modeling – ER*, Vol. 7532 LNCS. Springer Berlin Heidelberg, Berlin, Heidelberg, 397–408. https://doi.org/10.1007/978-3-642-34002-4_31

[2] David Benavides, Pablo Trinidad, and Antonio Ruiz-Cortés. 2005. Automated Reasoning on Feature Models. In *Advanced Information Systems Engineering*. Springer, 491–503. https://doi.org/10.1007/11431855_34

[3] Thorsten Berger, Steven She, Rafael Lotufo, Andrzej Wąsowski, and Krzysztof Czarnecki. 2013. A study of variability models and languages in the systems software domain. *IEEE Transactions on Software Engineering* 39 (2013), 1611–1640. https://doi.org/10.1109/TSE.2013.34

[4] Krzysztof Czarnecki, Paul Grünbacher, Rick Rabiser, Klaus Schmid, and Andrzej Wąsowski. 2012. Cool Features and Tough Decisions: A Comparison of Variability Modeling Approaches. In *Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems*. ACM, NY, USA, 173–182. https://doi.org/10.1145/2110147.2110167

[5] Krzysztof Czarnecki, Simon Helsen, and Ulrich Eisenecker. 2005. Formalizing cardinality-based feature models and their specialization. *Software Process: Improvement and Practice* 10, 1 (jan 2005), 7–29. https://doi.org/10.1002/spip.213

[6] 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 (mar 2011), 77–114. https://doi.org/10.1007/s10515-010-0076-6

[7] Deepak Dhungana, Patrick Heymans, and Rick Rabiser. 2010. A Formal Semantics for Decision-oriented Variability Modeling with DOPLER. In *Fourth International Workshop on Variability Modelling of Software-Intensive Systems, Linz, Austria, January 27-29, 2010. Proceedings*. 29–35.

[8] Cosmin Dumitrescu, Patrick Tessier, Camille Salinesi, Sebastien Gérard, Alain Dauron, and Raul Mazo. 2014. Capturing Variability in Model Based Systems Engineering. In *Complex Systems Design & Management*. Springer International Publishing, 125–139. https://doi.org/10.1007/978-3-319-02812-5_10

[9] Holger Eichelberger and Klaus Schmid. 2015. Mapping the Design-space of Textual Variability Modeling Languages: A Refined Analysis. *Int. J. Softw. Tools Technol. Transf.* 17, 5 (Oct. 2015), 559–584. https://doi.org/10.1007/s10009-014-0362-x

[10] José A. Galindo, Deepak Dhungana, Rick Rabiser, David Benavides, Goetz Botterweck, and Paul Grünbacher. 2015. Supporting distributed product configuration by integrating heterogeneous variability modeling approaches. *Information and Software Technology* 62 (2015), 78 – 100. https://doi.org/10.1016/j.infsof.2015.02.002

[11] D. Harel and B. Rumpe. 2004. Meaningful modeling: what's the semantics of "semantics"? *Computer* 37, 10 (oct 2004), 64–72. https://doi.org/10.1109/MC.2004.172

[12] Øystein Haugen. [n. d.]. Common variability language (CVL) - OMG revised submission. OMGdocumentad/2012-08-05,2012

[13] Kyo Kang, Sholom Cohen, James Hess, William Novak, and A. Peterson. 1990. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. Technical Report. Software Engineering Institute, Carnegie Mellon University.

[14] Ahmet Serkan Karataş, Halit Oğuztüzün, and Ali Doğru. 2013. From extended feature models to constraint logic programming. *Science of Computer Programming* 78, 12 (dec 2013), 2295–2312. https://doi.org/10.1016/j.scico.2012.06.004

[15] Charles W. Krueger. 2007. BigLever Software Gears and the 3-tiered SPL Methodology. In *Companion to the 22Nd ACM SIGPLAN Conference on Object-oriented Programming Systems and Applications Companion (OOPSLA '07)*. ACM, New York, USA, 844–845. https://doi.org/10.1145/1297846.1297918

[16] Raúl Mazo, Paul Grünbacher, Wolfgang Heider, Rick Rabiser, Camille Salinesi, and Daniel Diaz. 2011. Using constraint programming to verify DOPLER variability models. In *Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems - VaMoS '11*. ACM Press, New York, USA, 97–103. https://doi.org/10.1145/1944892.1944904

[17] Raúl Mazo, Camille Salinesi, Daniel Diaz, Olfa Djebbi, and Alberto Lora-Michiels. 2012. Constraints: The Heart of Domain and Application Engineering in the Product Lines Engineering Strategy. *International Journal of Information System Modeling and Design* 3, 2 (2012), 33–68. https://doi.org/10.4018/jismd.2012040102

[18] Raphael Michel, Andreas Classen, Arnaud Hubaux, and Quentin Boucher. 2011. A Formal Semantics for Feature Cardinalities in Feature Diagrams. In *Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems (VaMoS '11)*. ACM, New York, USA, 82–89. https://doi.org/10.1145/1944892.1944902

[19] Juan C. Muñoz-Fernández, Gabriel Tamura, Irina Raicu, Raúl Mazo, and Camille Salinesi. 2015. REFAS: a PLE approach for simulation of self-adaptive systems requirements. In *Proceedings of the 19th International Conference on Software Product Line - SPLC '15*. ACM Press, New York, USA, 121–125. https://doi.org/10.1145/2791060.2791102

[20] Klaus Pohl, Günter Böckle, and Frank van der Linden. 2005. *Software Product Line Engineering*. Springer Berlin Heidelberg, Berlin, Heidelberg. https://doi.org/10.1007/3-540-28901-1

[21] Clément Quinton, Daniel Romero, and Laurence Duchien. 2013. Cardinality-based feature models with constraints. In *Proceedings of the 17th International Software Product Line Conference on - SPLC '13*. ACM Press, New York, New York, USA, 162. https://doi.org/10.1145/2491627.2491638

[22] Fabricia Roos-Frantz, David Benavides, and Antonio Ruiz Cortés. 2010. Automated Analysis of Orthogonal Variability Models using Constraint Programming.. In *JISBD*. 269–280.

[23] Fabricia Roos-Frantz, David Benavides, Antonio Ruiz-Cortés, André Heuer, and Kim Lauenroth. 2012. Quality-aware analysis in product line engineering with the orthogonal variability model. *Software Quality Journal* 20, 3-4 (2012), 519–565.

[24] Camille Salinesi, Raul Mazo, Daniel Diaz, and Olfa Djebbi. 2010. Using Integer Constraint Solving in Reuse Based Requirements Engineering. In *Proceedings of the 2010 18th IEEE International Requirements Engineering Conference (RE '10)*. IEEE Computer Society, Washington, DC, USA, 243–251. https://doi.org/10.1109/RE.2010.36

[25] Klaus Schmid, Rick Rabiser, and Paul Grünbacher. 2011. A comparison of decision modeling approaches in product lines. In *Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems - VaMoS '11*. ACM Press, New York, USA, 119–126. https://doi.org/10.1145/1944892.1944907

[26] Pierre-Yves Schobbens, Patrick Heymans, Jean-Christophe Trigaux, and Yves Bontemps. 2007. Generic Semantics of Feature Diagrams. *Comput. Netw.* 51, 2 (2007), 456–479. https://doi.org/10.1016/j.comnet.2006.08.008

[27] Angela Villota, Raúl Mazo, and Camille Salinesi. 2018. On the Ontological Expressiveness of the High-Level Constraint Language for Product Line Specification. In *System Analysis and Modeling. Languages, Methods, and Tools for Systems Engineering*. Springer, Copenhagen, 46–66. https://doi.org/10.1007/978-3-030-01042-3_4

[28] Zippel and Contributors. [n. d.]. kconfig-language.txt. https://www.kernel.org/doc/Documentation/kbuild/kconfig-language.txt