A Systematic Analysis of Textual Variability Modeling Languages

Holger Eichelberger and Klaus Schmid

Software Systems Engineering, Institute of Computer Science,
University of Hildesheim, Germany0
{eichelberger, schmid}@sse.uni-hildesheim.de

Please cite this publication as follows:
Holger Eichelberger and Klaus Schmid. "A Systematic Analysis of Textual Variability Modeling Languages". In: *Proceedings of the 17th International Software Product Line Conference (SPLC'13)*. ACM, 2013, pp. 12–21. DOI: 10.1145/2491627.2491652.

The corresponding BibTeX-entry is:

```
@INPROCEEDINGS{EichelbergerSchmid13a,
  author = {Holger Eichelberger and Klaus Schmid},
  title = {A Systematic Analysis of Textual Variability Modeling Languages},
  booktitle = {Proceedings of the 17th International Software Product Line
               Conference (SPLC'13)},
  year = {2013},
  pages = {12--21},
  publisher = {ACM},
  doi = {10.1145/2491627.2491652},
}
```

Comment: Based on additional information we got from colleagues, we found that languages may have capabilities not initially represented in the publication. Moreover, after submission languages were extended or further information regarding capabilities was published, which is not captured in the publication. An up-to-date representation of language capabilities can be found at:
www.sse.uni-hildesheim.de/textual-variability-overview

A Systematic Analysis of Textual Variability Modeling Languages

Holger Eichelberger
University of Hildesheim
Marienburger Platz 22
31141 Hildesheim, Germany
+49 5121 883 762
eichelberger@sse.uni-hildesheim.de

Klaus Schmid
University of Hildesheim
Marienburger Platz 22
31141 Hildesheim, Germany
+49 5121 883 760
schmid@sse.uni-hildesheim.de

ABSTRACT

Industrial variability models tend to grow in size and complexity due to ever-increasing functionality and complexity of software systems. Some authors report on variability models specifying several thousands of variabilities. However, traditional variability modeling approaches do not seem to scale adequately to cope with size and complexity of such models. Recently, textual variability modeling languages have been advocated as one scalable solution.

In this paper, we provide a systematic analysis of the capabilities of current textual variability modeling languages, in particular regarding variability management in the large. Towards this aim, we define a classification schema consisting of five dimensions, classify ten different textual variability modeling languages using the classification schema and provide an analysis. In summary, some textual variability modeling languages go beyond textual representations of traditional variability modeling approaches and provide sophisticated modeling concepts and constraint languages. Three textual variability modeling approaches already support mechanisms for large-scale variability modeling such as model composition, modularization, or evolution support.

Categories and Subject Descriptors

D.2.1 [**Requirements/Specifications**]: *Languages*; D.2.13 [**Reusable Software**]: *Domain engineering*; D.3.3 [**Language Constructs and Features**]: *Constraints*;

General Terms

Languages, Design, Theory, Management.

Keywords

Software product lines, variability management, textual variability modeling, variability modeling in the large, scalability.

1. INTRODUCTION

Variability modeling is at the heart of software product line engineering [28, 34]. Many approaches to variability modeling have been developed over time. Currently, the most frequently used family of approaches in industrial practice is feature modeling [8], which was initially introduced in [27]. Further approaches include decision modeling [18, 44] or orthogonal variability modeling [34] and several others. Today, many variations of these basic approaches exist. For some of them the semantic equivalence of their modeling concepts has been shown, e.g., for free feature diagrams in [43] and for basic decision modeling and feature modeling in [20].

Currently, the focus of research in variability modeling is changing. As noted in [8, 11], size and complexity of variability models is increasing. Nowadays, industrial variability models may cover several thousand variabilities [8] or even up to more than 190.000 variabilities [11]. Thus, a new demanding question in variability modeling is *"Which variability modeling concepts are needed to support variability modeling in the large?"*

While graphical (tree-based) variability modeling approaches have a long tradition in product line engineering, they do not scale well in the context of such large models [10]. Some reasons stated in literature are that the creation of large industry-size variability models using a graphical language is difficult and requires dedicated tool support [10, 16], that some elements such as (textual) attributes and cross-tree constraints tend to clutter the layout [10, 16]. Also the (complete) layout of (feature) trees typically requires large physical dimensions, thus, impacting understandability [10]. This is one of the reasons why textual variability modeling languages increasingly attract attention. Further, textual variability modeling languages may also ease the integration into existing text-based or batch-oriented production processes and it is more easily possible to automatically create many different variants.

As part of this trend, recently, several textual variability modeling languages such as VSL [1], FAMILIAR [2], Clafer [4], TVL [16] and VELVET [38] have been invented to provide solutions to the issues of traditional graphical models in variability management. It seems, these languages (including one developed by of the authors) have been mostly created independently from each other. Thus, we believe that much can be gained from creating an overview on the most relevant existing textual variability modeling languages to study their conceptual approaches and to analyze their differences and commonalities in order to provide further input for the evolution of variability modeling languages. Besides providing the user of such technologies with a sound basis for selecting one of them over the other, the goal is also to inform future research, as strengths of one language can translate into improvement potential for another and common weaknesses may translate into future research challenges.

Therefore, we will target in this paper two research questions:

SPLC 2013, August 26 - 30 2013, Tokyo, Japan

http://dx.doi.org/10.1145/2491627.2491652

RQ1: Which modeling concepts are provided by current textual variability modeling languages? Or from a different point of view: Are textual variability modeling languages just a textual representation of traditional variability modeling approaches?

RQ2: Which concepts do current textual variability modeling languages support for variability modeling in the large? As discussed above, several textual variability modeling languages were invented to overcome the issues of graphical variability modeling languages, in particular regarding their support for large-size variability [10, 16]. Thus, we are interested in the specific concepts provided by textual variability modeling languages to support models containing thousands of variabilities.

The contribution of this paper is a systematic analysis of ten current textual variability modeling languages. Our analysis will help in collecting and structuring the current knowledge about textual variability modeling languages. As surveys or comparisons about such languages are missing, the underlying knowledge is currently distributed over several publications. We will summarize this knowledge and based on our analysis, our contribution will provide an overview and fill this gap in terms of answering RQ1 and RQ2.

This paper is structured as follows. In the next section, we will discuss related work. In Section 3, we will introduce our classification schema. In Section 4, we will briefly introduce the ten textual variability modeling languages and their individual background. In Section 5, we will then compare these ten variability modeling languages according to our classification schema. In Section 6, we will summarize and discuss the results of the comparison. Finally, in Section 7, we will conclude.

2. RELATED WORK

Several surveys and analyses of variability modeling have been published so far. *Focused analyses* compare variability modeling languages within one family of modeling approaches, while *broad analyses* target multiple families. We will first discuss focused analyses and then broad analysis. However, as we will show below, so far there is no specific comparison of textual variability modeling languages.

Focused analyses compare approaches within one family. We will start with focused analyses on traditional variability modeling and then continue with work on textual languages in variability management. Classen et al. [17] compare the definition of the term "feature" in eight different feature modeling approaches, but the authors do not analyze the specific modeling capabilities of the approaches. Schobbens et al. [43] survey feature diagram variants to derive a common syntax as well as a common formal semantics. Schmid et al. [42] compare multiple aspects of five different decision modeling approaches, ranging from modeling capabilities to product derivation support. Benavides et al. [7] review feature modeling concepts of 42 different feature modeling approaches. Although these analyses are not specific to textual variability modeling, we will consider some of the aspects analyzed in [7, 42, 43] as a foundation for our classification schema in this paper.

Some work provides a focused analysis of textual languages in variability management. Classen et al. [16] briefly compare nine textual feature modeling languages as part of a related work discussion and include also languages which are not designed for human-readability such as the file format of pure::variants [9]. In this paper, we will not consider such approaches but, instead, we will include recent approaches to textual variability modeling. Günther et al. [24] compare languages for realizing variability (called configuration languages in [24]). Such languages specify how to turn a product configuration into structural and behavioral program modifications in order to realize individual variabilities. The authors analyze these configuration languages in seven dimensions. Although this work is not directly on (textual) variability modeling languages, it provides three interesting aspects for large-scale variability modeling, namely composition as applied to variability modeling in [19, 23, 39], extensibility [39] and modularity [39].

Broad analyses compare approaches from multiple families. Berger et al. [8] describe a recent empirical study on the use of variability modeling approaches in industry. Chen et al. [13] surveyed variability management approaches with respect to their evolution over time. Istoan et al. [25] classify variability modeling approaches on the meta-model level, in particular in terms of the different types of models and their use in product line engineering. However, all three studies do not discuss modeling concepts in detail. Czarnecki et al. [18] compare multiple aspects of decision modeling and feature modeling concepts in general including modeling concepts and semantic richness. We will also take the analyzed aspects in [18] into account as a basis for our classification schema.

In summary, several publications target the analysis, comparison and classification of variability modeling approaches. Most of them focus on traditional variability modeling, while only [16] compares textual variability modeling languages as a basis for a benchmark of a new textual language. Thus, according to our best knowledge, currently no other survey explicitly classifies and compares the modeling capabilities of textual variability modeling languages on a detailed level.

3. CLASSIFICATION SCHEMA

In this section, we introduce the schema we use for the classification of textual variability modeling languages. First, we define some basic concepts. Then, we use them to describe the individual dimensions of our classification schema.

The *problem space* contains elements describing *what* the systems in a given domain must do. The problem space can be characterized by activities of domain modeling. In contrast, the *solution space* details *how* the elements in the problem space can be realized, e.g., by an architecture model of a specific system. A variability model defines the *configuration space* of a software product line as a particular view on the problem space. The configuration space is defined by *configurable elements,* each representing an individual variability. A *configuration* specifies the resolutions of the configurable elements and, thus, defines the instantiation of the generic artifacts implementing the variabilities in the solution space. A configuration may be *complete*, i.e., specific values are assigned to all configurable elements, or *partial*, i.e., no resolution value is given for at least one configuration element (see also staged configurations in [20]). *Constraints* are expressions over configurable elements specifying the validity of configurations.

For deriving a classification schema, we decomposed the research questions them into individual aspects which operationalize the respective question. Basically, we aimed at one dimension of analysis aspects for each research question. However, the number of aspects we obtained for RQ1 was rather diverse so that we structured the aspects for RQ1 into three top-level dimensions. The fourth top-level dimension targets RQ2, while the last

dimension aims at collecting some general characteristics about the textual languages. Our five top-level dimensions are:

- Capabilities of configurable elements (RQ1)
- Constraint support (RQ1)
- Configuration support (RQ1)
- Variability modeling in the large (RQ2)
- Language characteristics

We will now detail these five top-level dimensions. As part of the individual discussions of the dimensions we will refer to those analysis aspects which are already mentioned in literature and subsumed by our classification schema.

D1. **Configurable elements:** This dimension characterizes the capabilities of configurable elements themselves.

- *Forms of variation [22, 43]:* We will characterize the basic capabilities of a textual variability modeling approach according to its support for forms of variation such as optional, alternative or multiple selection [7, 28, 34]. Further, we consider extension [22, 39], i.e., variability without a predefined range of possible variations. Extension is important in open-world scenarios such as service-based systems or ecosystem.
- *Unit of variability [18]:* This aspect characterizes the predominant modeling concept such as "feature" in feature-related languages.
- *Attached information:* Can additional information such as (feature) attributes [16], binding times [18, 43] or even arbitrary metadata be attached to a configurable element? Attached information can be used to provide different levels of additional information characterizing a customizable element.
- *Data types [18, 42]:* Similar to programming languages, a variability modeling language may be typed. Typically, some basic types are given such as Boolean or Integer. Further, the language may provide mechanisms for creating user-defined types such as enumerations or structs (groups of configurable elements of possibly different types) or containers (sets of unnamed configurable elements of the same type). However, we have to differentiate whether the customizable elements or the attached information is typed. Typed configurable elements can be used to represent the form of variation, such as an enumeration may be used to express an alternative selection. Also attached information may be typed.
- *Cardinalities [7, 16]*: Cardinality denotes how many instances of a configurable element can be part of a configuration. The cardinality may be specified explicitly, implicitly through the data type of a configurable element or even in terms of constraints. Thus, we will consider cardinality as an aspect in this as well as in the constraint support dimension.
- *References:* Does the language allow references among configurable elements? At first glance, references do not seem to be a big deal as they can be understood as aliases to configurable elements. However, dependent on the actual semantics, references can be used to share configurations or to even configure networks of configurable elements. As several textual languages provide the concept of references, we will explicitly address references as an analysis aspect and not as a (derived) datatype as this is for example done in [42].

D2. **Constraint support** [7, 16, 18, 42, 43]**:** Typically, a variability modeling language supports some form of constraints among customizable elements to specify valid configurations. However, the support for constraints may differ significantly ranging from simple excludes-/requires-dependencies [7], over propositional logic and propositional logic with relational expressions up to first-order logic or even temporal logic. In addition, constraints may contain (arithmetic) expressions to calculate values, specify details of cardinalities or define constraints on data types.

D3. **Configuration support:** Often, a variability modeling language focuses on the customizable elements and their interdependencies but not on specific configurations. However, a textual variability modeling language can provide means for specifying (partial or complete) configurations in an integrated way. Further, default values [11] can simplify the configuration process as an initial (default) value can be specified. Default values may be overridden in later stages of the configuration, e.g., by a redefinition or by the final configuration value. A language may also support the calculation of default values (from other default values).

D4. **Scalability support:** Modeling concepts for large-scale variability modeling is a current trend in software product line research. We will structure this dimension into

- *Composition*: Can configurable elements (preferably in terms of larger units such as models) be composed into a single model [19, 23, 39]? For example, this enables variability models for sub-systems to be composed into one system-level model. In particular, composition capabilities are beneficial in multi-product-lines [23].
- *Modularity [18]:* On which level of granularity (if at all) is modularity [39] supported? For example, modularity may be supported by information hiding such as defining an interface on configurable elements and hiding elements not stated in the interface.
- *Evolution*: Does the variability modeling language provide specific concepts to support evolution? One example of such concepts is explicit versioning [39].

D5. **Language characteristics:** This dimension characterizes additional properties of the language itself. We will discuss whether the language was inspired by some existing (programming) languages, whether there is a certain predominant structure, whether there are specific concepts for supporting the user during the configuration process as well as the degree of formality of the language definition.

4. TEXTUAL VARIABILITY MODELING LANGUAGES

In this section, we introduce ten languages for textual variability modeling described in literature. We identified these languages based on an intensive literature search using the following key words: "textual", "variability", "modeling" or "language". In order to have clearly defined inclusion and exclusion criteria we used the following ones:

- Designed for sole variability modeling by humans, i.e., configurable elements or constraints can be specified.
- Described in terms of a complete language specification.
- Described in scientific literature (articles, papers, technical reports).
- Interpretable in an automated way such as parsing, semantic analysis, etc.

For identifying relevant approaches in a comprehensive manner, we searched the ACM digital library, IEEE Explore and Scopus and used Google Scolar to cross-check for completeness. In summary, we obtained more than 17 approaches to textual variability modeling. Due to the criteria listed above, we selected ten for our analysis in this paper. We excluded the following approaches: Synthesis [44], the DSL-approach by Völter and Visser [52], the Component Definition Language (CDL) [50] and the XML-based approach by Cechticky et al. [12] due to incomplete or missing language specifications, the Variability Modeling Language (VML) [29] and CDL [50] due to their integration with architecture modeling, the XSchema-based approach by Zhou et al. [53] due to issues with human readability as well as KConfig [47] and eCOS [21] due to missing scientific publications detailing the variability modeling language itself.

Below, we will discuss the ten individual approaches, we selected, as well as their background in the order of their publication. We will compare these approaches in the next section according to our classification schema introduced in Section 3.

The **Feature Description Language (FDL)** [51] is a textual notation for feature diagrams. The language was constructed by applying design principles of Domain-Specific Languages (DSLs). In addition to the syntax, the authors specify a feature diagram algebra. This algebra is a set of rules for operating on FDL models such as normalization, expansion and satisfaction rules. Further, the authors describe the mapping of FDL models to UML class diagrams for Java code generation. It is interesting to note that the excluded language CDL [50] is very similar to FDL except for some specific types of (component) constraints.

The tree-grammar approach by **Batory** [6] represents cardinality-based feature diagrams using (iterative) tree grammars. In particular, Batory represents constraints in terms of propositional formulae. The approach is implemented in a tool called GUIDSL as part of the AHEAD tool suite [5]. GUIDSL provides reasoning capabilities by employing a truth maintenance system.

The **Variability Specification Language** (VSL) [1, 37] aims at the integration of prominent feature modeling approaches with configuration links and variable entities. In VSL, configuration links are a specific form of references. Variable entities enable the modeling of predefined chains and networks of configurations interrelated by configuration links. VSL is a part of the Compositional Variability Management framework (CVM) [1], which also provides graphical views on VSLs as feature diagrams.

The **Simple XML Feature Model** (SXFM) is a textual format for representing variability models rather than a variability modeling language. SXFM is used in the feature model repository Software Product Lines Online Tools (S.P.L.O.T) [30, 31]. Basically, the approach is similar to tree-grammars by Batory [6] as SXFM embeds a textual representation of the tree including Boolean constraints into XML elements.

FAMILIAR [2, 3] is a scripting language which aims at defining, combining, analyzing and manipulating feature models. Due to the scope of this paper, we will focus on the variability modeling concepts rather than on the scripting language capabilities. The Eclipse-based tools of FAMILIAR consist of a textual editor, an interpreter as well as various model format, reasoner, and graphical editor integrations.

The **Text-based Variability Language** (TVL) [10, 15, 16] is a textual feature modeling notation. TVL was developed based on the experience that graphical feature modeling is a barrier to industrial adoption from a psychological as well as from a rational point of view [16]. TVL is designed as a scalable and lightweight approach to textual feature modeling, including cardinality-based decomposition, feature attributes and modularization mechanisms. The reference implementation of TVL contains a parser and a reasoning library.

µTVL (micro TVL) [14] is a subset of TVL which aims at core feature modeling. Basically, µTVL drops some types and changes some semantics. Further, µTVL enables multiple feature trees in one model. The µTVL tool support is integrated into the Abstract Behavioral Specification (ABS) suite and a specific reasoner allows analyzing the satisfiability of µTVL variability models.

The **CLAss, FEature, Reference** approach (Clafer) [4] combines meta-modeling (of classes) with first-class support for feature modeling. Clafer aims at a minimal number of concepts with a uniform semantics. Clafer provides specialization and extension layers via constraints and inheritance, explicit containment, cardinalities, multiple instances and (object) references. Tool support for Clafer consists of a parser, a semantic analyzer, a code generator for Alloy [26] and a reasoning mechanism.

VELVET [38] aims at support for separation of multi-dimensional concerns in feature-based variability modeling. Some basic concepts in VELVET are inspired by TVL. VELVET provides several mechanisms to combine the individual models, i.e., concerns, into a common variability model. The tool support for VELVET includes a parser, a dependency analyzer as well as format conversion tools, e.g., to transform SXFM into VELVET.

The **INDENICA Variability Modeling Language** (IVML) [36] is developed in the EU-funded project INDENICA [35] on customization and integration of service platforms. IVML is designed as a scalable, textual variability modeling language, which supports the variability modeling requirements that are relevant to support the customization of complex service platform ecosystems [22, 40]. These requirements include non-Boolean variability, complex dependencies, large-scale variability, quality of service constraints and support for software ecosystems. Although these requirements may imply a specialized variability language for service-based systems, IVML was designed as a general-purpose variability modeling language based on a decision modeling approach [41]. IVML is also integrated into the Eclipse-based EASy-Producer Tool [23].

5. LANGUAGE CLASSIFICATION

In this section, we classify the textual variability modeling languages described in the last section according to our classification schema. We will structure this section according to the dimensions of our schema introduced in Section 3 and summarize the results in Section 6.

5.1 Configurable Elements

Support for modeling configurable elements is the first dimension (D1) of our classification schema. In this section, we will discuss

	Forms of variation	Unit of variability	Attached information	Data types				Cardinalities	Refer-ences
				predefined	derived	user-defined	applies to		
FDL	o,a,m	feature	-	-	-	-	-	(x)	-
Batory	o,a, (m)	feature	-	-	-	-	-	feature	-
VSL	o,a, (m), e	feature, group	parameter, user attribute	Boolean, Integer, Float, String	-	enum	parameter, user attribute	feature, group	x
SXFM	o,a, m	feature, group	-	-	-	-	-	group	-
FAMILIAR	o,a,m	feature, group	-	Boolean, Integer, Real, String, enum, model, configuration	container	-	feature, model, configuration	-	(x)
TVL	o,a,(m),e	feature, group	attribute, data block	Boolean, Integer, Real	-	enum, struct, constant	attribute	group	x
µTVL	o,a,(m),e	feature, group	attribute	Boolean, Integer	-	-	attribute	group	-
Clafer	o,a,(m),e	clafer	feature	Integer, String	-	-	feature	clafer, feature	x
VELVET	(o),a,(m), e	concept, feature	attribute	Boolean, Integer, Float, String	-	-	attribute	(group)	-
IVML	o,a,m,e	decision variable	meta-attribute	Boolean, Integer, Real, String	container, typedef	enum, compound	decision variable, meta-attribute	x	x

Table 1: Properties of the configurable elements.
(o=optional, a=alternative, m=multiple selection, e=extension, x=supported, (x)=implicitly supported, -=not supported)

the support for the individual analysis aspects of D1. Table 1 summarizes the results of the discussion below. In Table 1, as well as in the following summary tables, the approaches are listed in the sequence given in Section 4.

FDL offers explicit concepts for optional, alternative and multiple selection of features. In FDL, cardinalities can be specified in terms of keywords such as "one-of" or "more-of" implying certain fixed lower and upper limits. However, FDL does neither support data types nor attached information nor references.

The tree-grammar approach by **Batory** allows expressing optional and alternative selection of features. Cardinalities can be stated in terms of lower and upper bounds and, thus, implicitly allow multiple selection. Similar to FDL, the approach by Batory does not support attached information, types, or references.

Basically, **VSL** follows the terminology of cardinality-based feature modeling, i.e., units of variation are features and feature groups. VSL supports optional and alternative selection as forms of variation. Further, multiple selection is implicitly supported via cardinalities. Regarding attached information, VSL allows typed parameterized features as well as additional user-specific information in terms of typed attributes. VSL defines four basic types and supports user-defined enumerations. Finally, VSL defines a specific form of references, namely configuration references, which are links between VSL models and their configurable elements to allow the configuration of features depending on the configuration given in the referenced model.

SXFM is rather similar to the tree-grammar approach by Batory. In addition, SXFM supports feature groups with numeric cardinalities.

FAMILIAR allows modeling optional, alternative and mandatory features, but no cardinalities. Types in FAMILIAR are predefined, i.e., no user-defined types can be created. Types are distinguished into primitive and complex types, whereas complex types are accessed through references (akin to Java). Types and, thus, applicable operations, are determined at runtime. Results of operations can be stored in (typed) variables.

TVL provides explicit support for optional and alternative features. In addition, features can be extended by an arbitrary number of refining features providing additional (typed) attributes. TVL provides three basic types as well as user-defined constants and types such as enumerations or structs. Attributes of type struct are treated as a specific type of reference (links). Cardinalities can only be attached to feature groups, which implicitly allow multiple selection. Data blocks may be used to store key-value-pairs of strings.

µTVL inherits several concepts from TVL but drops data blocks, the real type, user defined datatypes, and references.

A **Clafer** model consists of type definitions and constraints. Type definitions may contain multiple typed features or may be abstract. Clafer supports optional and alternative selection, (implicit) multiple selection as well as extension as forms of variation. In Clafer, alternatives can be expressed as extended types. The basic types are numeric (Integer) and String. Cardinalities can be attached to groups as well as to features. Clafer supports references in order to express shared instances. Containment features are a unique capability of Clafer expressing the ownership of individual (unshared) instances.

The top-level element of a **VELVET** model is the "concept", which acts as the root of a feature tree. A feature tree consists of features and feature groups. The forms of variation are (implicit) optional, alternative, and multiple selection, the latter through cardinalities. Features may further be specified by typed attributes using one of the predefined basic types. The feature tree in a "concept" may be extended in subsequent models. VELVET supports two types of group cardinalities, namely "someof" and "oneof" but no lower or upper limits can be stated explicitly.

Configurable elements in **IVML** are represented by a typed decision variable. IVML supports Boolean, numeric (Integer,

	constraint expressions					cardi-nalities	type restrict-tion
	basic	proposi-tional	first-order	relati-onal	arith-metic		
FDL	x	-	-	-	-	-	-
Batory	-	x	-	-	-	-	-
VSL	x	(x)	(x)	-	x	-	x
SXFM	-	(x)	-	-	-	-	-
FAMILIAR	-	x	-	-	-	-	-
TVL	-	x	-	x	x	-	x
µTVL	x	x	-	x	x	-	-
Clafer	-	x	(x)	x	-	-	-
VELVET	-	x	-	x	-	-	-
IVML	-	x	x	x	x	x	x

Table 2: Constraint capabilities
(x=supported, (x)=partially supported, -=not supported)

Real) and String as basic types. The model can define derived types such as typed container (set, sequence) as well as (constrained) type aliases. In addition, enumerations as well as compounds of decision variables can be specified. Due to these types, IVML supports optional, alternative and multiple selection as well as extension (of compounds) as form of variation. Further, each decision variable can be attributed by user-defined typed variables (meta-attributes), e.g., to represent binding times. Meta-attributes are equipped with the same expressiveness as decision variables and may, in particular, be used in constraints. In IVML, cardinalities are implicitly represented by the types of the decision variables and may be further restricted by constraints. IVML also supports references, i.e., aliases to shared decision variables.

Although we did not explicitly consider decomposition [16, 18, 43] as an analysis aspect, it is interesting to note that all approaches provide some kind of decomposition. Feature-based approaches rely on the feature decomposition hierarchy, Clafer on containment and IVML on the nesting of compounds.

5.2 Constraint support

Constraint support is the second dimension (D2) in our classification schema and characterizes the capabilities of textual variability modeling languages in defining restrictions for valid configurations. Table 2 summarizes the results of the discussion below. We discuss the supported type of logic, whether arithmetic expressions can be used in constraints, whether constraints can be used to specify cardinalities or even dependencies among cardinalities and whether types can be restricted using constraints.

FDL supports only basic dependencies such as required, excluded, or included features.

The constraint capabilities in **Batory's approach** rely on propositional logic. The tree-grammars in this approach are implicitly mapped to propositional constraints by production rules in order to validate optional and alternative selections. Thus, propositional constraints are supported for cross-tree constraints.

VSL provides basic dependencies (needs, excludes, alternatives, suggests, impedes), propositional formulae without implications and arithmetic expressions in constraints. The value range of types can be restricted. In case of numeric types, interval expressions on constant values can be specified, i.e., VSL supports a specific form of relational expressions. In case of String parameters, regular expressions can be specified for type restriction.

SXFM supports propositional formulae without implications.

FAMILIAR supports propositional formulae including equality operators on content and references.

TVL, µTVL and **VELVET** provide constraints in terms of propositional formulae with relational expressions for numeric attributes. Further, µTVL offers basic dependencies such as require or exclude as well as arithmetic calculations in constraints. TVL and µTVL support inclusion and exclusion expressions on feature trees. In addition, TVL provides predefined aggregation functions such as calculating the sum of a certain attribute for all features in the feature tree as well as domain expressions for individual attributes.

Clafer supports a specific form of first-order logic. In Clafer, so called default quantifiers such as "some" can be used in combination with propositional expressions and relational expressions for numeric features. Clafer constraints are similar to Alloy constraints [26].

Basically, the constraint language of **IVML** is a variant of the Object Constraint Language (OCL) [32] with additions for default values and value assignments. Thus, IVML also supports first-order logic on containers in combination with propositional, relational, and arithmetic expressions. Further, IVML supports the specification of cardinalities (a refinement of the implicit cardinalities discussed in Section 5.10), e.g., by limiting the sizes of containers, and even dependencies among cardinalities. Finally, IVML supports the restriction of existing types using constraints.

5.3 Configuration support

In addition to the specification of configurable elements, their properties and dependencies, a variability modeling language may provide concepts for defining configurations. Basically, this may happen in terms of default values for configuration elements or by value assignments in constraints. Further, a (textual) variability modeling language may also provide specific support for defining partial or complete configurations in an integrated way.

Batory's approach and **SXFM** do not provide any specific support for configurations.

FDL supports default values given in terms of constants.

FAMILIAR enables the explicit creation of configurations for a given feature model and for a specific configuration the selection, deselection and unselection of features. Although not explicitly stated in literature, we expect that partial configurations can be defined in FAMILIAR.

TVL, µTVL, VELVET and **Clafer** support only value assignments. However, these languages do not provide explicit concepts for value assignments but rely on constraints which implicitly fix attribute or feature values, respectively.

VSL and IVML provide sophisticated capabilities for defining configurations in an integrated way. **VSL** defines a specific

	default values	value assignment	configuration	
			partial	complete
FDL	x	-	-	-
Batory	-	-	-	-
VSL	x	x	x	x
SXFM	-	-	-	-
FAMILIAR	-	x	(x)	x
TVL	-	(x)	-	-
µTVL	-	(x)	-	x
Clafer	-	(x)	-	-
VELVET	-	(x)	x	x
IVML	x	x	x	x

Table 3: Configuration support
(x=supported, (x)=partially supported, -=not supported)

concept for configurations which allows fixing the values and the specific cardinalities for individual features. In VSL, existing configurations may be referred and reused in terms of configuration networks. As maintaining such configuration networks may become a tedious process, VSL aims at simplifying this by so called variable entities, i.e., predefined chains and networks of feature configurations. In contrast, **IVML** does not explicitly distinguish between variability model and (partial) configuration. In IVML, configurable elements are grouped at top-level in "projects". A project may either contain only type definitions, decision variables and default values, i.e., a variability model, or value assignments, i.e., a configuration, or mix these concepts to realize partial, staged and extensible configurations.

5.4 Scalability support

Specifying several thousands of variabilities in a single variability model is similar to realizing a large object-oriented system in one class. Specific concepts such as those used in current programming paradigms are needed to cope with the challenges of large variability models. In this section, we will classify the ten textual variability modeling approaches with respect to their support for fundamental concepts of large product lines (D4), namely composition [19, 23, 39], evolution [42] and modularity [42]. Table 5 summarizes the results.

The historical first approaches in our comparison, namely **FDL**, **Batory's approach** and **SXFM** do not provide mechanisms for variability modeling in the large. **Clafer** supports composition on type level (containment features) but not on model level.

VSL provides mechanisms for composition and modularity. VSL provides public and private visibility for elements in a variable entity, i.e., for feature models, configurations and references. Further, VSL enables the composition of variable entities. However, no explicit support for evolution is described in [1, 37].

FAMILIAR provides two composition operators, the "merge" operator for overlapping and the "aggregate" operator for disjoint models. In FAMILIAR, modules can be defined on script level (including contained feature models) and individual configurable elements can be exported or hidden.

In **TVL** and **µTVL**, models can be included, i.e., composed to a model of larger scale.

VELVET aims at multi-dimensional separation of concerns in variability modeling. For creating models containing the individual concerns, it enables to import the namespace of another model and provides composition in terms of multiple inheritance of "concepts" (including the feature trees), superimposition akin to aspect-orientation and aggregation of configurations.

	related language	comments	predominant structure	specification
FDL	-	-	tree	grammar
Batory	-	-	tree	grammar
VSL	C, Java	//, /* */	graph	examples
SXFM	XML		tree	examples
FAMILIAR	FeatureIDE, (Java)	-	variables, tree	(grammar), (formal)
TVL	C	//	graph	formal
µTVL	TVL	-	tree	formal
Clafer	Alloy	--	declarations, nesting	examples
VELVET	TVL	-	tree	grammar
IVML	Java, OCL	//, /* */	declarations, nesting	grammar

Table 4: Language characteristics (-=not supported)

	composition	evolution	modularity
FDL	-	-	-
Batory	-	-	-
VSL	variable entities	-	model, configuration
SXFM	-	-	namespace
FAMILIAR	merge, aggregate	-	-
TVL	include	-	-
µTVL	include	-	-
Clafer	-	-	-
VELVET	inheritance, superimposition, aggregation	-	-
IVML	import	model version	interface

Table 5: Capabilities for large-scale modeling (-=not supported)

IVML also supports to import existing models, i.e., variability models, configurations and mixed models in terms of their namespaces. To support evolution, imports in IVML may be restricted to a certain range of versions of the referenced model. This enables the independent evolution of IVML models, while keeping the consistency of existing compositions. Regarding modularity, IVML provides the concept of interfaces. An IVML project may specify multiple alternative named interfaces, which export the listed decision variables.

5.5 Language characteristics

All variability modeling languages analyzed in this paper are textual languages. Some of the languages share further commonalities. In this section, we will discuss some of these commonalities such as related (programming) languages explicitly mentioned as a source of inspiration in the related publications, the style of comments, the predominant structure of the models as well as the degree of formality of the language definition (examples, grammar, formal semantics). Table 4 summarizes the results except for user support, as none of the languages provide specific concepts in this regard.

FDL and **Batory's** approach are structured according to (feature) trees. However, both languages do neither support comments nor provide user support mechanisms. Both languages are specified by their respective grammar.

FAMILIAR is organized in terms of trees (feature models) and variables containing models or derived results. The language is largely inspired by FeatureIDE [49] as well as Java (regarding references). In literature, FAMILIAR is mostly described by examples, some operators are given in formal semantics and the grammar is stated in the online manual [45].

VSL, **SXFM, TVL, µTVL** and **VELVET** models are organized as trees. Due to references, VSL and TVL models may form a graph. The SXFM format mostly consists of a tree-grammar embedded into an XML element. The containing XML document contains also the cross-tree constraints. VSL and TVL share concepts with C or Java in order to reduce learning efforts. Consequently, VSL and TVL support C/Java-like comments. µTVL is derived from TVL and VELVET inherits some concepts from TVL but, however, neither the µTVL grammar in [14] nor the VELVET grammar in [48] specify a syntax for comments. While VSL and SXFM are described in terms of examples, VELVET is specified by its grammar and TVL as well as µTVL by their grammar and a formal semantics.

Clafer is a combination of feature- and meta-modeling. Clafer models are basically organized by type declarations. Although not explicitly stated in [4], type declarations in Clafer look akin to object-oriented programming languages. The constraint notation of Clafer is modeled after Alloy [26]. Also the comments follow the Alloy syntax. Clafer is described in [4] in terms of examples.

The syntax of the decision variables and type declarations in **IVML** is inspired by Java while the IVML constraint language shares several similarities with OCL [32]. Comments in IVML are stated in Java syntax. Variable and type declarations (nested in a project) determine the structure of an IVML model. While IVML models themselves do not provide explicit user support, the IVML tooling enables internationalized descriptions for decision variables. IVML is specified by examples and its grammar in [46].

6. DISCUSSION

In this section, we discuss the results of our classification. In the last section we collected conceptual details of the individual languages and already summarized them in overview tables. In this section, we will identify common concepts and differences which will finally help answering the research questions of this paper in Section 7. We will structure the discussion according to the dimensions of our classification schema and summarize the conclusion for each individual dimension.

Configurable elements (D1): All textual variability modeling languages support optional, alternative and multiple selection as forms of variation. However, in one language (VELVET), optional selection is the implicit default. Further, in most languages multiple selection is only implicitly available in terms of cardinalities. Recently developed languages support also (different ways for) the extension of already defined configurable elements. Eight approaches rely on feature modeling terminology. Clafer combines feature modeling with meta-modeling. IVML is based on a decision modeling approach. Seven approaches support the specification of cardinalities, two of them using fixed keywords (FDL, VELVET), six via specification of lower and upper level boundaries and one (IVML) via typed configuration elements and constraints. Five approaches support references among elements using rather specific semantics.

Seven approaches implement a type system. Basic types include Boolean, typically Integer, sometimes real-valued types (named either Real or Float) or String. Three languages support user-defined enumerations, two user-defined structs. Five approaches support attached information, mostly in the sense of typed feature attributes. Only IVML uses types to characterize configurable elements themselves so that types indicate the form of variation. Two languages (VSL and IVML) enable some orthogonal type of attached information. This is used in VSL to capture user-specific information while in IVML such meta-attributes can be used in (nearly) arbitrary ways and can even be used in constraints.

We conclude that current textual variability modeling languages provide all the typical concepts of traditional (graphical) variability modeling. Some languages also provide a type system, (typed) attached information or even references among configurable elements and thus go beyond typical capabilities.

Constraint support (D2): Nine textual variability modeling languages allow constraint specifications according to propositional logic. Out of these nine approaches, five support relational expressions (for numerical types) and two first-order logic. Four approaches allow arithmetic expressions in constraints, and three provide concepts for type restrictions using constraints. In summary, five out of all approaches rely on non-Boolean logic, which implies specific challenges for reasoning.

We can summarize that current textual variability modeling languages support rather complex constraint syntaxes and, thus, most of them cover the typical constraints in variability modeling.

Configuration support (D3): Three approaches support default values to simplify the configuration. However, default values impose further challenges for reasoning mechanisms as (dependent on the semantics of the language) defaults may be overridden and, thus, lead to non-monotonic reasoning, i.e., changing default values may invalidate previous conclusions.

Seven approaches support value assignments to define the specific values in a configuration, either as part of constraints or as individual statements. However, only five approaches provide explicit configuration concepts and three of them explicitly support partial / staged configuration.

Variability modeling in the large (D4): Six approaches provide concepts for composing a model from other models. VALVET offers three different mechanisms for composition, FAMILIAR two, while the remaining four approaches provide exactly one (default) composition mechanism. Only IVML allows restricting compositions based on explicit version numbers to support evolution. Three approaches provide mechanisms for modularity.

In summary, only three of the recent textual variability modeling languages face the challenge of large-scale variability modeling.

Language characteristics (D5): Eight approaches rely on (feature) trees as predominant structure. The remaining two are structured in terms of sequences of type declarations (of Clafers or decision variables) or variables (FAMILIAR). Four approaches support comments. Three of these approaches are inspired by Java or C in order to simplify the uptake in practice. No approach provides user support for the configuration process. Three languages are described in terms of examples, five also by their grammar and three (TVL and μTVL, FAMILIAR for selected operations) using formal semantics.

It seems that most approaches analyzed in this paper were developed in isolation, while taking over basic concepts from traditional variability modeling. Three approaches form a new family, namely TVL, VELVET and μTVL, as the latter two derive concepts to some extent from TVL. Further, most languages are (indirectly) inspired by well known programming languages.

7. CONCLUSION

Size and complexity of variability models in industry is growing. Some literature reports about variability models with thousands of variables. The highest number of variabilities in a model we know of so far is close to 200.000 configurable elements [11]. Thus, variability modeling approaches must be scalable to effectively support such models. Textual variability modeling languages are one promising approach to provide scalability and understandability. The core idea is to create models in a similar way as software systems are programmed today. However, to cope with such large models, also appropriate concepts for composition and modularization must be provided.

In this paper, we analyzed ten textual variability modeling approaches. We used a classification schema with five dimensions, namely concepts for customizable elements, support for constraints, configuration support, concepts for large-scale variability modeling and general language characteristics. Based on our analysis, we can answer the research questions as follows:

RQ1: Basically all approaches support the modeling concepts and constraints required for traditional variability modeling. In particular, this is true for those approaches which explicitly represent cardinality-based feature modeling in textual form such as FDL, Batory's approach or SXFM. More recent approaches developed since 2009 such as FAMILIAR, VSL, the TVL family, Clafer or IVML provide advanced variability and constraint modeling capabilities such as configuration references, a type system or non-Boolean constraints.

RQ2: Currently, only three approaches (FAMILIAR, VSL and IVML) provide mechanisms to face the challenge of large models. Both provide concepts for modularity including information hiding and composition. Only IVML provides evolution support (version-constrained composition).

In this paper, we analyzed and compared the capabilities of ten textual variability modeling languages on a conceptual level. As a result of this analysis we answered both research questions. Further, we contributed to the current knowledge by reviewing and systematically organizing existing but so far distributed knowledge about textual variability modeling languages. As a more general result of this analysis we see the trend that (textual) variability modeling languages tend to provide even more sophisticated and complex concepts such as type systems, constraints or capabilities for variability modeling in the large. This will lead to new challenges, in particular regarding analyzing and reasoning such models.

We believe that this analysis will provide interesting input to researchers in the area and will help them to further improve their variability modeling techniques. While we believe that our conceptual analysis already provides good data, for the future it would be interesting to let researchers and tool vendors apply their approaches to a common set of modeling problems, e.g., in a contest such as CoCoMe [33], to analyze the individual expressiveness, scalability, advantages and disadvantages.

8. ACKNOWLEDGEMENTS

This work was partially supported by the INDENICA project, funded by the European Commission grant 257483, area Internet of Services, Software & Virtualisation (ICT-2009.1.2) in the 7th framework programme.

9. REFERENCES

[1] A. Abele, Y. Papadopoulos, D. Servat, M. Törngren, and M. Weber. The CVM Framework - A Prototype Tool for Compositional Variability Management. In *WS on Variability Modelling of Software-Intensive Systems (VAMOS'10)*, pages 101–105, 2010.

[2] M. Acher, P. Collet, P. Lahire, and R. France. FAMILIAR: A Domain-Specific Language for Large Scale Management of Feature Models. *Sci. Comput. Programm.*, 78:657–681, 2013.

[3] M. Acher, P. Collet, P. Lahire, and R. B. France. Composing feature models. In *SLE*, pages 62–81, 2009.

[4] K. Bak, K. Czarnecki, and A. Wasowski. Feature and Meta-models in Clafer: Mixed, Specialized, and Coupled. In *Conf. on Software Language Engineering (SLE '10)*, pages 102–122, 2010.

[5] D. Batory. AHEAD Tool Suite. http://www.cs.utexas.edu/users/schwartz/ATS.html [validated: February 2013].

[6] D. Batory. Feature Models, Grammars, and Propositional Formulas. In *Software Product Line Conference (SPLC'05)*, pages 7–20. 2005.

[7] D. Benavides, S. Segura, and A. Ruiz-Cortes. Automated Analysis of Feature Models 20 Years Later: A Literature Review. *Information Systems*, 35:615–636, 2010.

[8] T. Berger, R. Rublack, D. Nair, J. Atlee, M. Becker, K. Czarnecki, and A. Wasowski. A Survey of Variability Modeling in Industrial Practice. In *WS on Variability Modelling of Software-intensive Systems (VaMoS '13)*, pages 1–8, 2013.

[9] D. Beuche. Modeling and Building Software Product Lines with Pure::Variants. In *Software Product Line Conference (SPLC '08)*, page 358, 2008.

[10] Q. Boucher, A. Classen, P. Faber, and P. Heymans. Introducing TVL, a Text-based Feature Modelling Language. In *WS on Variability Modelling of Software-intensive Systems (VaMoS'10)*, pages 159–162, 2010.

[11] H. Brummermann, M. Keunecke, and K. Schmid. Formalizing Distributed Evolution of Variability in Information System Ecosystems. In *WS on Variability Modeling of Software-Intensive Systems (VaMoS '12)*, pages 11–19, 2012.

[12] V. Cechticky, A. Pasetti, O. Rohlik, and W. Schaufelberger. Xml-based feature modelling. In *Conf. Software Reuse (ICSR '04)*, pages 101–114, 2004.

[13] L. Chen, M. Ali Babar, and N. Ali. Variability Management in Software Product Lines: A Systematic Review. In *Software Product Line Conference (SPLC '09)*, pages 81–90, 2009.

[14] D. Clarke, R. Muschevici, J. Proença, I. Schaefer, and R. Schlatte. Variability Modelling in the ABS Language. In *Symposium on Formal Methods for Components and Objects (FMCO'10)*, pages 204–224, 2010.

[15] A. Classen, Q. Boucher, P. Faber, and P. Heymans. The TVL Specification. Technical Report P-CS-TR SPLBT-00000003, PReCISE Research Center, University of Namur, 2010.

[16] A. Classen, Q. Boucher, and P. Heymans. A text-based approach to feature modelling: Syntax and semantics of TVL. *Sci. Comput. Program.*, 76:1130–1143, 2011.

[17] A. Classen, P. Heymans, and P.-Y. Schobbens. What's in a Feature: A Requirements Engineering Perspective. In *Conf. on Fundamental Approaches to Software Engineering (FASE'08)*, 2008.

[18] K. Czarnecki, P. Grünbacher, R. Rabiser, K. Schmid, and A. Wasowski. Cool Features and Tough Decisions: A Comparison of Variability Modeling Approaches. In *WS on Variability Modelling of Software-Intensive Systems (VaMoS '12)*, pages 173–182, 2012.

[19] K. Czarnecki, S. Helsen, and U. Eisenecker. Staged Configuration Using Feature Models. In *Software Product Line Conference (SPLC'04)*, pages 162–164, 2004.

[20] K. Czarnecki, S. Helsen, and U. Eisenecker. Staged configuration through specialization and multi-level configuration of feature models. *Software Process Improvement and Practice*, 10(2):143–169, 2005.

[21] eCos Team. eCos Home Page. http://ecos.sourceware.org/ [validated: March 2013].

[22] H. Eichelberger, C. Kröher, and K. Schmid. Variability in Service-Oriented Systems: An Analysis of Existing Approaches. In *Conf. on Service-Oriented Computing (ICSOC'12)*, pages 516–524, 2012.

[23] S. El-Sharkawy, C. Kröher, and K. Schmid. Supporting Heterogenous Compositional Multi Software Product Lines. In *Software Product Line Conference (SPLC '11)*, vol. 2, 2011.

[24] S. Günther, T. Cleenewerck, and V. Jonckers. Software variability: the design space of configuration languages. In *WS on Variability Modeling of Software-Intensive Systems (VaMoS '12)*, pages 157–164, 2012.

[25] P. Istoan, J. Klein, G. Perouin, and J.-M. Jezequel. A Metamodel-based Classification of Variability Modeling Approaches. In *VARY WS: VARiability for You*, 2011.

[26] D. Jackson, I. Schechter, and H. Shlyahter. Alcoa: The Alloy Constraint Analyzer. In *Conf. on Software Engineering (ICSE'00)*, pages 730–733, 2000.

[27] K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21 ESD-90-TR-222, 1990.

[28] F. van der Linden, K. Schmid, and E. Rommes. *Software Product Lines in Action - The Best Industrial Practice in Product Line Engineering*. Springer, 2007.

[29] N. Loughran, P. Sánchez, A. Garcia, and L. Fuentes. Language support for managing variability in architectural models. In *Software Composition*, pages 36–51. 2008.

[30] M. Mendoca. *Efficient Reasoning Techniques for Large Scale Feature Models*. PhD thesis, University of Waterloo, Ontario, Canada, 2009.

[31] M. Mendonca, M. Branco, and D. Cowan. S.P.L.O.T.: Software Product Lines Online Tools. In *Conf. Object oriented programming systems languages and applications (OOPSLA '09)*, pages 761–762, 2009.

[32] Object Management Group, Inc. (OMG). Object Constraint Language. Specification v2.00 2006-05-01, Object Management Group, 2006. Available online at: http://www.omg.org/docs/formal/06-05-01.pdf.

[33] Dagstuhl Modelling Contest Organizers. CoCoMe - The Common Component Modelling Example. http://www.cocome.org/ [validated: March 2013].

[34] K. Pohl, G. Böckle, and F. van der Linden. *Software Product Line Engineering: Foundations, Principles, and Techniques*. Springer, 2005.

[35] INDENICA project consortium. INDENICA Research Project Website, 2010. http://www.indenica.eu [validated: February 2013].

[36] INDENICA project consortium. Open Variability Modeling Approach for Service Ecosystems. Technical Report Deliverable D2.1, 2012. http://www.indenica.eu [validated: February 2013].

[37] M.-O. Reiser. Core Concepts of the Compositional Variability Management Framework (CVM) – A Practitioner's Guide. Technical Report 2009-16, Technische Universität Berlin, 2009.

[38] M. Rosenmüller, N. Siegmund, T. Thüm, and G. Saake. Multi-dimensional variability modeling. In *WS on Variability Modelling of Software-Intensive System*, pages 11–20, 2011.

[39] K. Schmid. Variability Modeling for Distributed Development - A Comparison with Established Practice. In *Software Product Lines: Going Beyond*, pages 151–165. 2010.

[40] K. Schmid. Variability Support for Variability-Rich Software Ecosystems. In *Product LinE Approaches in Software Engineering (PLEASE'13)*, 2013.

[41] K. Schmid and I. John. A customizable approach to full-life cycle variability management. *Sci. Comput. Programm.*, 53(3):259–284, 2004.

[42] K. Schmid, R. Rabiser, and P. Grünbacher. A Comparison of Decision Modeling Approaches in Product Lines. In *WS on Variability Modeling of Software-Intensive Systems (VaMoS '11)*, pages 119–126, 2011.

[43] P.-Y. Schobbens, P. Heymans, and J.-C. Trigaux. Feature Diagrams: A Survey and a Formal Semantics. In *Requirements Engineering Conf.)*, pages 139–148, 2006.

[44] Software Productivity Consortium Services Corporation, Technical Report SPC-92019-CMC. *Reuse-Driven Software Processes Guidebook, Version 02.00.03*, 1993.

[45] FAMILIAR team. Familiar reference manual. https://nyx.unice.fr/projects/familiar/wiki/manual [validated: March 2013].

[46] IVML team. IVML language specification. http://projects.sse.uni-hildesheim.de/easy/docs/ivml_spec.pdf [validated: March 2013].

[47] Linux Kernel Team. KConfig. http://www.kernel.org/doc/Documentation/kbuild/kconfig-language.txt [validated: February 2013].

[48] The MultiPLe team. The Velvet Modeling Language. http://fosd.de/multiple/ [validated: March 2013].

[49] T. Thüm, C. Kästner, F. Benduhn, J. Meinicke, G. Saake, and T. Leich. FeatureIDE: An extensible framework for feature-oriented software development. *Sci. Comput. Programm.*, 2012.

[50] T. van der Storm. Variability and component composition. In *Conf. Software Reuse (ICSR '04)*, pages 157–166, 2004.

[51] A. van Deursen and P. Klint. Domain-Specific Language Design Requires Feature Descriptions. *Journal of Computing and Information Technology*, 10:1–17, 2002.

[52] M. Voelter and E. Visser. Product Line Engineering using Domain-Specific Languages. In *Software Product Line Conference (SPLC '11)*, pages 70–79, 2011.

[53] J. Zhou, D. Zhao, L. Xu, and J. Liu. Do we need another textual language for feature modeling? In *Software Engineering Research, Management and Applications 2012*, volume 430 of *Studies in Computational Intelligence*, pages 97–111. 2012.