Modeling Variability in the Video Domain: Language and Experience Report

Mauricio Alférez, Mathieu Acher, José A Galindo, Benoit Baudry, David Benavides

Software Quality Journal manuscript No.
(will be inserted by the editor)

Modeling Variability in the Video Domain: Language and Experience Report

Mauricio Alférez · Mathieu Acher · José A. Galindo · Benoit Baudry · David Benavides

the date of receipt and acceptance should be inserted later

Abstract [**Context**] In an industrial project, we addressed the challenge of developing a software-based video generator such that consumers and providers of video processing algorithms can benchmark them on a wide range of video variants.

[**Objective**] This article aims to report on our positive experience in modeling, controlling, and implementing software variability in the video domain.

[**Method**] We describe how we have designed and developed a variability modeling language, called VM, resulting from the close collaboration with industrial partners during two years. We expose the specific requirements and advanced variability constructs we developed and used to characterize and derive variations of video sequences.

[**Results**] The results of our experiments and industrial experience show that our solution is effective to model complex variability information and supports the synthesis of hundreds of realistic video variants.

[**Conclusions**] From the software language perspective, we learned that basic variability mechanisms are useful but not enough; attributes and multi-features are of prior importance; meta-information and specific constructs are relevant for scalable and purposeful reasoning over variability models. From the video domain and software perspective, we report on the practical benefits of a variability approach. With more automation and control, practitioners can now envision benchmarking video algorithms over large, diverse, controlled, yet realistic datasets (videos that mimic real recorded videos) – something impossible at the beginning of the project.

Keywords variability modeling · feature modeling · software product line engineering · configuration · automated reasoning · domain-specific languages · video testing

Mauricio Alferez
SnT Centre for Security, Reliability and Trust,
University of Luxembourg, Luxembourg
E-mail: alferez@svv.lu

Mathieu Acher
DiverSE team,
Univ Rennes, Inria, CNRS, IRISA, France
E-mail: mathieu.acher@irisa.fr

José A. Galindo
Department of Computer Languages and Systems,
University of Seville, Spain
E-mail: galindo@us.es

Benoit Baudry
KTH, Royal Institute of Technology
E-mail: baudry@kth.se

David Benavides
Department of Computer Languages and Systems,
University of Seville, Spain
E-mail: benavides@us.es

1 Introduction

Numerous organizations manage *variability*, encoded as features or configuration options, to extend, change, customize or configure multiple kinds of artifacts (e.g., hardware devices, operating systems or user interfaces) [1,2]. The delivery of hundreds of variants promises to achieve substantial profit in terms of customer satisfaction, mass customization, or market presence. Variability techniques have been successfully applied in many domains such as automotive, avionics, printers, mobile, or operating systems [3–5]. However, different domains and applications pose specific challenges to variability engineering, both in terms of modeling language and implementation. Practitioners need empirically-tested techniques and languages for efficiently modeling and implementing variability in a systematic and scalable manner. Berger et al. [6] warn that the lack of experience reports on variability modeling techniques may impede the progress of variability research. The goal of this article is precisely to report on the use of variability techniques in the video domain and in an industrial setting.

Specifically, in a project involving providers and consumers of vision algorithms, we faced the challenge of synthesizing video sequence variants with software. This challenge is very original given the kind of artifact that varies (videos) and its domain (video analysis). The current practice to obtain videos is to collect or to film them. The participants of the project pointed out that the current practice is not economically viable or sufficient for testing video analyzers. We give more specific details in Section 2, but essentially, high costs and complex logistics are required to (1) film videos in real locations and (2) annotate videos with the expected results (ground truths). The difficulties of the current practice to obtain an input set of testable videos brought our attention. In this article, we report on our experience applying a variability-based solution to the problem. The objective of our variability approach was first; i) *to automate* the synthesis of large datasets (video variants) corresponding to a combination of features and attributes (configurations); and ii) *to control* the synthesis process (selecting the characteristics of the desired videos). This prevents the derivation of invalid or unrealistic video variants (e.g., some features of the video are known to be incompatible); covers a large diversity of video variants to test algorithms in different, varying settings, and allows practitioners to scope the synthesis (select a subset of video variants) and specify the kinds of video variants they want to obtain.

This article aims to report insights on how variability modeling techniques have been used in an industrial setting. Throughout different iterations, we integrated multiple constructs – existing in the literature and implemented in different variability languages – into a single solution. We also introduce some new constructs to ease off the reasoning capabilities required for sampling the set of configurations used as test-suite. The resulting solution is a new *language*, called VM[1], for modeling variability of videos and achieving the synthesis of a set of video variants. In the remainder of the article, we use the terms "feature" and "attribute" to refer to Boolean or numerical information that describe the characteristics of a video (e.g., the luminance level, whether there are distractors, vehicles, etc.). The combinations of features and attributes' values are called configurations and are exploited to synthesize concrete variants of videos.

We show that Boolean variability constructs, though useful, are clearly not sufficient in the video domain as it has to cope with numeric parameters (also called *attributes*) and features appearing several times (also called *clones* [7] or *multi-features* [8]). For example, *speed* is an attribute of a vehicle that can take different values across videos, and *vehicle* is a multi-feature as each video can show several vehicles configured with different speed values. In addition to the support of attributes and multi-features, we describe several language constructs of VM. For example: attribute default values, deltas to discretize continuous domain values, objective functions to filter relevant configurations, multi-ranges for attributes domains, meta-information, etc. Furthermore, VM provides additional constructs for having an effect on the automated reasoning about configurations. Reasoning has multiple interests [9]; it can be used to sample valid configurations, to interactively configure a system (in different steps or stages), to find the "best" configuration, etc. The need of reasoning capabilities has two additional motivations in our context. First, video experts can better *control* the way configurations are generated – precluding irrelevant videos while ensuring the coverage of relevant testing scenarios. Second, users can proactively reduce the computational time of *automated reasoning* operations in charge of sampling valid configurations. Such reasoning is based on solving techniques and may be expensive since numerous combinations of values are possible; we provide language mechanisms to scale up the reasoning.

This article is a significant extension of a paper published at ISSTA conference in 2014 [10]. Our early work [10] contributed in the software testing field, but did neither describe nor evaluate our variability language. It did not report on insights of our overall variability solution either. We provide here in-depth

[1] VM stands for Variability Modeling. In essence, VM is a variability language and shares many properties of e.g., feature modeling languages. We chose the acronym VM since it may also stand for Video Modeling, hence the word play.

details of our close collaboration (with the industrial partners during two years) that has lead to the design of VM. We also describe the role of the VM language when developing the video generator we demonstrated at SPLC conference in 2014 [11].

This article addresses the lack of solid evaluation of the impact of variability models on the industry [12, 6,13–15]. The novel constructs of VM have been discussed, validated, and developed with participants of the project during meetings and technical development. Finally, we highlight the benefits of a variability approach in the video domain. Our recent development [16] further increased the quality of the video generator and was possible because of proper variability management with VM. We are now capable of synthesizing hundreds of video variants – something impossible at the beginning of the project – opening avenues for large-scale benchmarking of video algorithms.

This article addresses the following research questions: What are the practical considerations of applying our variability language VM **(RQ1)**? How effective are VM language constructs for reasoning **(RQ2)**? What are the practical benefits of a variability-based approach **(RQ3)**? What are the commonalities and differences between constructs of VM and state-of-the-art variability languages **(RQ4)**? We hope the description of VM and the experience report can help researchers and practitioners in the application of variability modeling techniques and languages.

In summary, this article provides the following contributions:

- A description of *VM, a variability language* resulting from the close collaboration with industrial partners. We expose the basic and advanced variability constructs we have to develop and use to characterize variations of video sequences;
- The description of *VM constructs that help automated reasoning operations* for generating configurations (with features and attributes) in an efficient and controlled way;
- *Analytic and empirical results* that show that VM language and the reasoning support fulfill the requirements of the project as well as the *lessons learned* when designing VM (Including a comparison with other existing variability languages);
- A discussion on the *benefits of a variability approach* and an improvement of current practice: with more automation and control, practitioners in the video domain can now envision to benchmark algorithms over large, controlled, diverse, yet realistic datasets.

The remainder of this paper is organized as follows. Section 2 further describes the industrial problem of video generation, reviews previous attempts, and outlines our variability-based approach. Section 3 describes the research method. Section 4 reports on the three main iterations for designing VM. Section 5 introduces VM including advanced mechanisms to address the specific requirements we faced for efficiently generating purposeful configurations. Section 6 evaluates our variability approach around four axes: (1) practical considerations for applying feature models and VM, (2) empirical results w.r.t. the adequacy and effects of the new constructs of VM (3) benefits of variability and improvements of practice (4) comparison of VM to other existing variability languages. Section 7 discusses the threats to validity of our results. Section 8 reviews other related work. Section 9 presents concluding remarks and summarizes the lessons learned.

2 Industrial Problem and Overview of the Solution

In this section, we introduce the industrial problem and the limitations of current practice. We then briefly describe our variability-based solution. The goal of this section is to provide an *overview* of both the problem and the solution. Research methods, elicitation of requirements, details about the variability solution, and validation of our contributions are provided in the next sections.

2.1 An Industrial Testing Problem

Video analysis systems are ubiquitous and crucial in modern society [17,18]. Their applications range from video protection, crisis monitoring, to crowds' analysis. *Video sequences* (videos in short) are acquired, processed and analyzed to produce numerical or symbolic information. The corresponding information typically raises alerts to human observers in case of interesting situations or events.

Depending on the goal of video sequence recognition, signal processing algorithms are assembled in different ways. Each algorithm is a complex piece of software, specialized in a specific task (e.g., segmentation, object recognition, tracking). Even for a specific task, a one-size-fits-all algorithm, capable of being efficient and accurate in all settings, is unlikely. The engineering of video sequence analysis systems, therefore, requires to choose and configure the right combination of algorithms [18].

The MOTIV project aims to improve the evaluation of computer vision algorithms such as those used for surveillance or rescue operations. Two companies are part of the MOTIV project as well as the DGA (the French governmental organization for defense procurement). The two companies develop and provide algorithms for video analysis. A targeted scenario is usually as follows. First, airborne or land-based cameras capture on-the-fly videos. Then, a video processing chain analyzes videos to detect and track objects, for example, survivors in a natural disaster. Based on that information the military personnel triggers a rescue mission quickly based on the video analysis information. The DGA typically consumes the video algorithms of the two companies for implementing the processing chains.

The diversity of scenarios and signal qualities poses a difficult problem for all the partners of MOTIV: which algorithms are best suited given a specific application? From the consumer side (the DGA), how to choose, select and combine the algorithms? From the provider side (the two companies), how to guarantee that the algorithms are appropriate for the targeted scenarios and robust to varying situations?

In practice, the engineering of such systems is an iterative process in which algorithms are combined and tested on various kinds of inputs (video sequences). Practitioners can eventually determine what algorithms are likely to fail or excel under certain conditions before the actual deployment in realistic settings such as using those algorithms in rescue operations. Admittedly, practitioners rely on empirical and statistical methods, based on numerous metrics (e.g., precision, recall, accuracy). Also, the major barrier remains to find a *suitable, comprehensive input set of video sequences for testing analysis algorithms*.

2.2 Actual Practice and Early Attempts

The current testing practice is rather manual, very costly in time and resources needed, without any qualitative assurance (e.g., test coverage) of the inputs (e.g., see [19,20]). Specifically, our partners need to collect videos to test their video analysis solutions and detection algorithms. They estimated that an input data set of 153000 videos (of 3 minutes each) would correspond to 320 days of video footage and requires 64 years of filming outdoors (working 2 hours a day). These numbers were calculated at the starting point of the project, based on the previous experiences of the partners. Moreover videos themselves are not sufficient; video practitioners need also to annotate videos in order to specify the expected results (i.e., *ground truths*) of video algorithms. This activity increases the amount of time and effort as well.

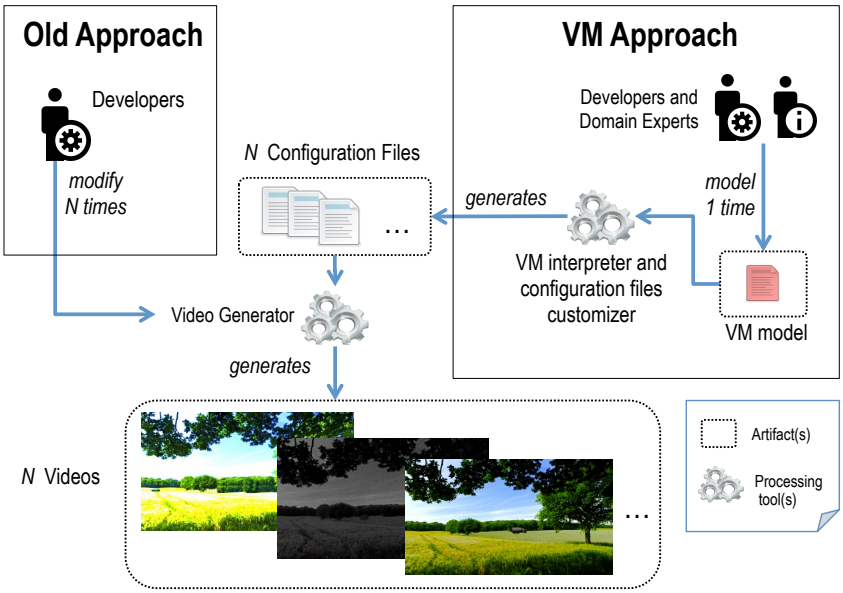

Figure 1 Old process compared to the VM-based process for video generation

Another possible approach is to modify or transform existing videos. The first attempt was therefore to create a video generator for producing customized videos – based on user preferences that were hard-coded during the first versions. It was realized by the industrial partners of the project. The left-hand side of Figure 1 shows this *old (generative) approach*. For deriving a variant, our partners had to manually comment

lines or modify variable values directly in the video generator code to change the physical properties and objects that appear in each video.

When the video generator was more mature, the partners together with us decided to create configuration files to communicate input values instead of scattering the parameters in different source code files. In particular, they employed Lua configuration files that have a simple structure based the pattern *parameter = value*. Then, developers used Lua code and proprietary C++ libraries, developed by a MOTIV partner, to process those configuration files and execute algorithms to alter, add, remove or substitute elements in base videos. Lua is a widely used programming language (`http://www.lua.org/`). Details about the computer vision algorithms that synthesize video variants are out of the scope of the paper. It should be noted that the generator not only computes a video variant but also a *ground truth* (or gold standard). A ground truth records the expected results and what objects of the video scenes should be recognized and tracked. It is used to assess the performance of video algorithms (accuracy, precision, recall, etc.). The automated synthesis of a ground truth is an important motivation behind the development of a video generator, since the manual specification of expected results is a labour-intensive and very costly process.

Despite the availability of a video generator, the effort remains tedious, undisciplined, and manual. It was still hard to construct large datasets – our partners have to manually modify the configuration file, with the additional problem of setting non conflicting values. Also they ignore what kinds of situations are covered or not by the set of videos, i.e., some kinds of videos may not be included in the dataset. Overall, more *automation* and *control* were needed to synthesize video variants in order to cover a large diversity of testing scenarios.

2.3 Variability Modeling Approach: An Overview

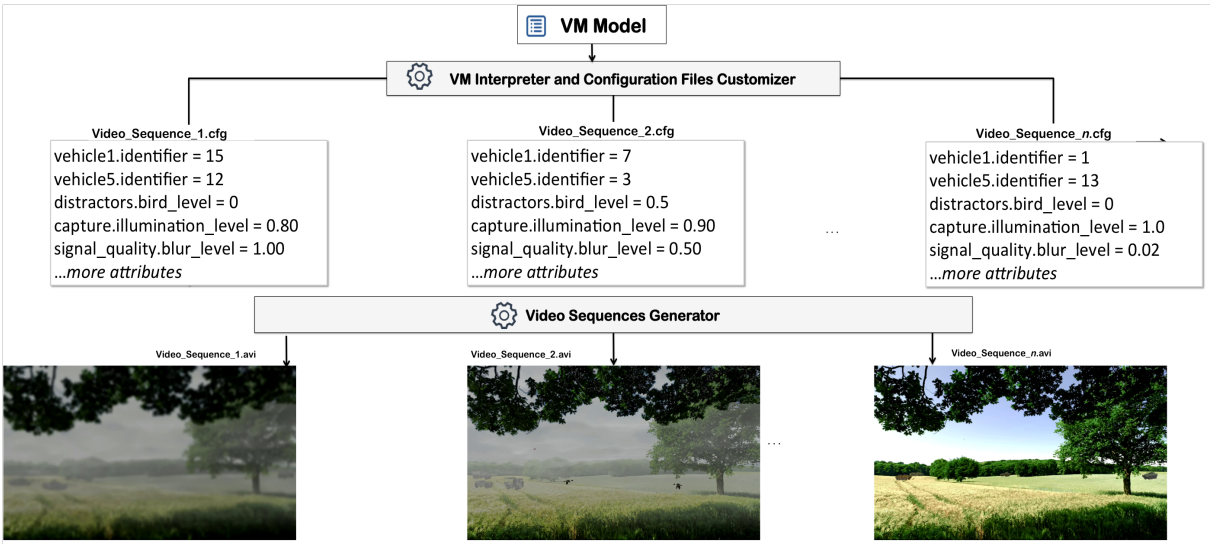

Figure 2 Three configurations files, generated from a VM model, and the three corresponding video variants

To overcome previous limitations, we introduced a variability-based approach (see right-hand side of Figure 1). The key idea is that practitioners now explicitly model variability using the VM a variability language we have developed in the project. This approach was developed from scratch but clearly inspired with: First, FaMa[21] and FAMILIAR[22] languages as because they were developed by members of the project. Second, by other more recent variability modeling languages such as Clafer[23, 14]. Using the VM language, variability is typically expressed in terms of mandatory, optional, mutually exclusive features, but also attributes for encoding non-Boolean values (integers, floats or strings). As detailed in the remainder of the paper, numerous other advanced constructs can be used for describing what can vary in a video.

A VM model is an abstraction of all possible Lua configuration files. It has the merit (1) of enforcing constraints over attributes and values (precluding invalid configurations); (2) reasoning techniques (e.g., constraint programming or satisfiability techniques) can operate over the model to assign values to features and

attributes in an efficient and sound way; (3) generative techniques can process the model to automatically produce configuration files.

Specifically we developed reasoning operations for producing configuration sets that cover the t-wise[2] criteria (while handling constraints and some objective functions over attributes).

Overall we can generate Lua configuration files (from a VM model) that the video generator can exploit to produce numerous video variants. Figure 2 provides an excerpt of three configuration files, each coming from a VM model. The three video variants (see bottom of Figure 2) correspond to the three configuration files. The level of illumination, the distractors, or the blur levels have three different values and are examples of what can vary in a video.

With this last approach and relying on the *Constraint Satisfaction Problem (CSP)* formulation presented in [10], we have been able to generate two sets of diverse configurations. Specifically, we generated a first set that introduces a lot of noises and moving objects for a very simple algorithm that detects everything moving in the scene by comparing two photograms in a row. The second set was developed for challenging tracking algorithms to keep the trajectory of a vehicle that crosses with another in the scene. For this second generation, the set of videos maximizes the number of concurrent vehicles in the scene as well as the number of occultants.

The generation of configurations was done locally, on a single computer. However, we distributed the video generations into the french grid computing Grid'5000 (`http://www.grid5000.fr`), since the synthesis of a video variant corresponding to one configuration takes between 30 minutes and 1 hour. Overall we have been able to synthesize 300Gb of videos representing around 3958 different video sequences. The resulting datasets were possible to obtain with a variability approach. The next sections will describe the introduction of variability in the project, the different iterations for meeting the requirements, and the validation of our work.

3 Research Method

Before going into details of VM (see Section 5), reasoning mechanisms (see Section 5.3) and an evaluation of the variability-based solution (see Section 6), we now describe how we conducted our research.

Our method is based on *applied research* (also called technology research) in which the objective is to create artifacts that are better in some manner than those already developed [25]. That is, the new artifacts improve the previous ones in speed, safeness or any other technological characteristic. Once the new artifacts are designed and developed, researchers have to show that the artifacts are complete with regards to the original requirements. Specifically, we follow the main steps (as in applied research [25]):

- *Problem statement:* researchers look for the potential need of a new technology or new artifacts. In our case it was the identification of the need of variability-based techniques for generating variants of videos;
- *Contribution:* in this step researchers actually develop a solution and create the artifacts supporting the solution. We developed throughout different iterations a variability language and a video generator;
- *Validation:* in this final step, researchers verify that all the requirements required to improve the previous solutions and, of course, to grant the solution of the problem have been fulfilled. We contrasted our results with the practitioners so they agree with the proposed solution.

Figure 3 provides a retrospective of the project with all major steps of our research method (in yellow), releases of the video generator (in red), iterations over the design of VM (in green), and workshops/technical meetings (in blue).

Workshops and technical meetings. We employed project meetings to elicit requirements. During a period of about 18 months, we participated in:

- *five large meetings*: each meeting consisted in a one-day workshop with all the partners. The duration of each workshop was 6 hours and all participants were involved. In average, participants were 2 researchers and 6 industrials. During the first meeting, we mainly learned about the problem of testing algorithms, hearing the difficulties faced by both sides (consumers and providers of video algorithms); we also explained variability principles and techniques. We then presented VM (see hereafter for more details about the iterations), solicited and gathered feedback, and presented major evolutions at each meeting. The last meeting was used to validate the realisticness of video variants;

[2] t-wise methods are widely used in combinatorial testing. Concretely, t-wise methods reduce the number of tests to execute while covering the interactions of t input parameters in a system. Also, the most common value for t is two (pair-wise), which was demonstrated to be valid in a wide range of scenarios [24]

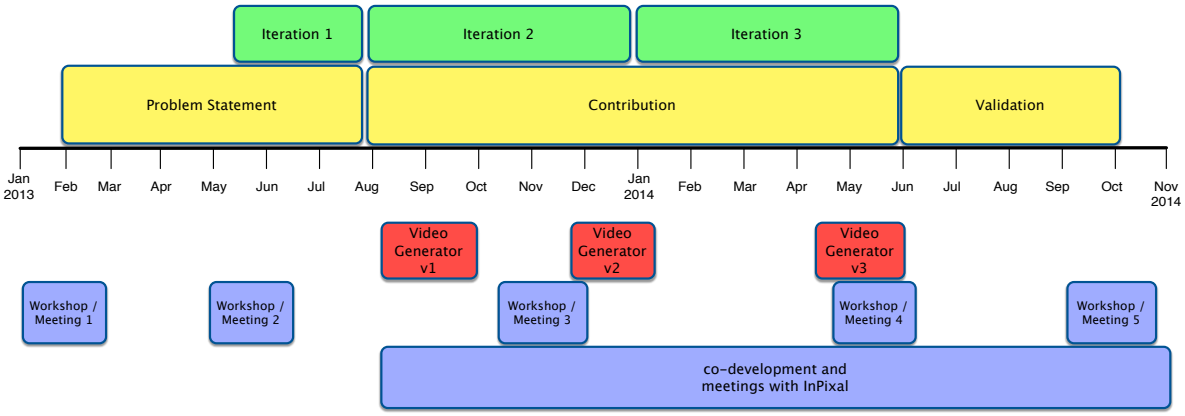

Figure 3 Retrospective of the MOTIV project

– *individual, physical meetings* with the main developer of the video generator (from InPixal company) for evolving the VM model in line with the Lua configuration files. Typically, the duration of such meetings was 3 hours while 2 researchers and 1 industrial were involved;
– *regular conversations* with participants by email and phone (around one per week depending on availability).

The introduction of variability techniques in an industrial setting was not straightforward and rather an iterative process. We incorporated some agile techniques regarding the meetings and workshops; we demonstrate VM and the video generator for gathering feedback and validating the different iterations. Two main problems came across and were addressed by:

– designing of a new language (VM) that offers adequate constructs for modeling variability information (e.g., attributes on continuous domains) in the video generator;
– developing of mechanisms on top of VM for controlling the automated and efficient generation of configurations.

Co-design of language and generator. VM was created in three main iterations. Each one adds more expressiveness to the language and led to the revision of the tooling support. The discussions and feedback we had during the workshops influenced the design of VM, but not only. The technical realization of variability (i.e., in the solution space and Lua code) was another important driving force. The reason is that the configurations generated from a VM model have to be compatible with the Lua generator. For each of the iteration, the connection with the realization layer – through configuration files – thus validated the design and evolution of VM. It explains why we released three versions of the video generator (see Figure 3) during the three iterations of VM.

4 Requirements and Iterations of the Variability Language

We now specifically describe the three iterations that lead to the design of VM (see Figure 3). Each iteration added more constructs to the language and had influence on the reasoning support as well. In this section, we specify the concrete requirements we obtained from the different rounds of meetings.

The first iteration included basic variability modeling (features, feature packages, and relationships between features). The second iteration included extended variability modeling (attributes and advanced relationships between features, such as multi-features and cardinality-based groups). Finally, the third iteration included extra variability modeling (model information, descriptions, and annotations). The detailed requirements of each iteration are described in a research report [26]. In this paper, we concentrate in the novel parts and in the realization of the requirements in VM.

We employed project meetings to elicit requirements. During a period of about 18 months, we had five large meetings, three individual meetings with the main developer of the video generator, and regular conversations with participants by emails. Currently, the main direct user of VM in the MOTIV project is a team of three people from Inria, France.

The design of VM has been influenced by the technical realization (i.e., the solution space) of variability. We jointly developed an end-to-end solution to generate configurations that are then fed to a video generator

(see Figure 4, page 16). The connection with the realization layer – through configuration files – validates the adequacy of VM throughout the different iterations.

4.1 First iteration: Categorization of features and basic organization of elements

Here we present the first very basic concern practitioners first had when the project started.

R1. Modeling characteristics of videos. In the first meeting the practitioners considered with interest the idea of explicitly modeling characteristics of a video. They quickly mention the need to organize and represent groups of *"probably excluding or requiring"* some characteristics. We presented the notion of feature diagram and they mostly agree with all the constructs it provide by default.

R2. Constraint support. In the same meeting some practitioners asked if we can describe requirements and exclude relationships between features being in different groups. Specifically, they were proposing the example of a video that is recorded in an urban scenario and, because of that, vehicles do not leave dust behind them. Several examples of potential constraints were given.

4.2 Second iteration: continuous and discrete variables encoding

R3. Modeling numerical and repetitive information. In the second iteration, we obtained a first version model that provides the configuration capabilities of a video generator. However, the practitioner in charge of generating videos told us that this approach was clearly not enough because many characteristics (or "parameters") are not Boolean, but rather numeric. Moreover, we cannot control how much a characteristic may appear in the scene with only a true or false value. Here we identified the need of encoding some quantitative properties such as the dust level behind a car or the number of vehicles that appear in a scene, each vehicle having its own characteristics.

R4. Documentation, maintainability and readability. During a long meeting, we ended by having a set of complex models, subject to different modifications, and elaborated by the experts from our initial model. Because of the growing complexity, different participants proposed and asked to have constructs for documenting the characteristics, values, constraints, etc. For instance, for improving the readability/maintenance of the VM model, participants wanted to describe the intent and justify the presence of an attribute like cost (see Listing 5, page 12).

R5. Dealing with complex constraints. The addition of attributes and repetitive characteristics prompts the need to define more complex constraints that complement the basic "requires" and "excludes" constraints between features defined in basic variability modeling. In particular, practitioners wanted to specify new kinds of constraints that involve features, attributes, and multi-features like "the selection of a countryside background implies to include less than 10 different men along the video" (see Listing 6, page 13).

R6. Bounded domain values and precision. Bounded attributes reduce the number of possible values of an attribute, and therefore, the number of combinations of attributes values and features. However, even bounded attributes may have a value among an almost infinite range and it is necessary to specify which values are the most important. For example, the bounded attribute "real man.appearance_change [0.0..1.0]" includes many and too close values (e.g., 0.00011 and 0.00012) that are not differentiated by the human eye and it makes no sense to produce two different videos that vary only on those values. Overall, there is the need of encoding the precision of parameters involved in the generation of a video.

4.3 Third iteration: Scaling up variability analysis

For the third workshop and after several small meetings with our coworkers, we ended by having a proposal to express the variability encoded in a video-sequence and we were able to synthesize some basic videos. However, we identified some lacks.

R7. Controlling the sampling of configurations. An observation is that too much configurations (and videos) can be generated from a VM model. It has two consequences. First, some of the videos' characteristics are simply not relevant for the targeted scenarios in which video algorithms are executed. Second, as we cannot enumerate and generate all videos, there is a need to *sample* configurations. Overall, domain experts need to control the generation process in a fine-grained way. The use of advanced constraints (see R5) is a possible solution, but is not sufficient. Specifically, experts wanted to maximize or minimize some (combinations of) parameters' values. For example, experts need to maximize the global illumination of

MOTIV requirements	VM constructs
R1. Modeling characteristics of videos	basic variability modeling (mandatory/optional features, feature groups, hierarchy, etc)
R2. Constraint support	cardinality-based groups and cross-tree constraints
R3. Modeling numerical and repetitive information	attributes (real, int), multi-features
R4. Documentation, maintainability and readability	feature hierarchy, model information and description, packages
R5. Dealing with complex constraints	constraints between features/attributes, wildcards
R6. Bounded domain values and precision	attribute domain, attribute types (real/int), delta, multi-delta
R7. Controlling the sampling of configurations	delta, multi-delta, default values, NT and ND, objective functions
R8. Differentiate static and run-time variabilities	annotations over features and attributes

Table 1 Requirements of MOTIV project and related VM constructs

the video and several parameters contribute to the illumination. In terms of sampling, we learned that the variations of *some* video characteristics/parameters' values are not necessary in some contexts, e.g., their values can be fixed. As a result, the sampling process can focus on the variations of other characteristics or parameters for diversifying the set of videos.

R8. Differentiating static and run-time variabilities. Some configuration options refer to changes in a video that will remain "as is" until the end of the video (static variability). However, there are other changes that are applied all along a video (run-time variability). For example, the speed of a vehicle can change throughout a video sequence and is a run-time variability (see Listing 3, page 11). The explicit distinction between these two kinds of variability is important to determine the binding time of each configuration option. This information is exploited to scope the spectrum of configurations sampling, i.e., to only those that have static variability.

5 Introducing VM

In this section, we describe the main constructs of VM, the variability language we have designed and developed throughout the MOTIV project. Table 1 represents the mapping relationship of the requirements proposed in Section 4 and the modeling constructs of VM.

Initially, we used an existing variability language (FAMILIAR) to model the problem of this project. However, when we arrived at the second iteration, we observed that some of constructs were missing and, when available, disseminated through different existing languages (see also Section 6.4). We found it difficult to reuse constructs and support scattered in different languages. Therefore, we decided to develop VM from scratch and have a complete flexibility (1) for introducing novel language constructs and defining the syntax; (2) for developing dedicated support (edition, reasoning, etc.)

VM is a textual, human readable language that supports the modeling of (extended) variability information. The objective of the VM language is to cope with the variability description needing existing in the video domain. However, VM offers generic variability mechanisms that are also suitable for more traditional variability[3]. We present here, in a nutshell, the basic and extended variability modeling capabilities of VM. Naturally, we use the video domain and the motivating scenario to illustrate VM. Readers can find the complete implementation code and grammar of VM online as well as the variability models we elaborated in the industrial project[4].

General language concepts. The elaboration of a VM model starts with the declaration of a package and some descriptions to document model elements. Then several blocks (relationships, attributes, constraints, etc.) are specified. Specifically, in the block Relationships, features are organized within a hierarchy together with variability information. In another block, attributes are specified with their types and domain values while annotations (like NT or ND), delta or default values can be added. We use the notion of feature and attributes for referring to characteristics of a video. Features have Boolean values while attributes express a numerical information. Attributes are associated with a feature since they describe an information related to a feature. Multi-features are features that can be duplicated (e.g., there could be more than 1 vehicle

[3] Some of the VM constructs are actually present in other variability languages while others are additional constructs or adaptations for modeling variability. Section 6.4 compares VM with other variability languages.

[4] `https://github.com/ViViD-DiverSE/VM-Source`

```
1  Relationships:
2  scene { //mandatory root feature
3    background { //mandatory feature
4      oneOf { // alternative feature group
5        urban  // grouped feature
6        countryside //grouped feature
7        desert //grouped feature
8      }
9    }
10   ? objects { //optional feature
11     someOf { //Or feature group
12       [1..5] vehicle  //short way to express multi-features
13       cloneBetween 1 and 10 man /*readable but verbose way to express multi-features. It is
             equivalent to [1..10] man*/
14     }
15   }
16 }
```

Listing 1 Feature relationships example

in a video). Finally, (complex) constraints and objective functions can be specified. The left-hand side of Figure 4, page 16 gives a comprehensive example. More details are given hereafter for each construct of VM.

5.1 Basic variability modeling

To satisfy the R1 requirement we proposed a set of basic constructs inspired from other traditional variability languages such as FaMa and FAMILIAR. VM defines configuration options as *features* and relate them in a block called "Relationships". This block, exemplified in Listing 1, shows a features' hierarchy where the selection of a feature (the *child* feature) depends on the selection of a more general feature (the *parent* feature). On the other side, logical dependencies are expressed using groups of alternative features where the selection of a grouped feature may be incompatible with the selection of other grouped features.

The different types of relationships available in VM are summarized next:

Root. Following traditional terminology from graph theory, a feature without a parent is called a *root* feature. In VM, each relationships block can have a root feature. "scene" (see Listing 1, line 2) is an example of a root feature.

Mandatory. Child feature is required. Corresponds to features that will be included in all possible video configurations such as "background" (see Listing 1, line 3).

Optional. Child feature is not required. This corresponds to features that may be or may not be selected as part of a video configuration. Optional features use the symbol "?" before their name, for example "? objects" (see Listing 1, line 10).

Alternative-group. One of the sub-features must be selected. An alternative-group is represented using the keyword "OneOf". For example, we specify in Lines 4-8 of Listing 1 that one can choose only one "background" between "urban", "countryside" and "desert".

Or group. At least one of the sub-features must be selected. An Or group is represented using the word "someOf". For example, we specify in lines 11-14 (again Listing 1) that "vehicle" and "man" are two not exclusive alternative kinds of objects that can be placed in a scene.

Cardinality-based groups. VM adds cardinalities to feature groups to specify the selection of a minimum and a maximum number of grouped features. The specification of a cardinality-based group follows one of the two patterns:
(1) `[`*`minVal`*`..`*`maxVal`*`]`, or
(2) `someBetween` *`minVal`* `and` *`maxVal`*. For instance, Listing 2 states that at least two features among distractors, occultants, specialClimateConditions should be selected while it is possible to select all three features.

This construct generalizes Alternative- and Or-group (e.g., the minimum and the maximum number of an Alternative-group is 1). It contributes to requirements R1 and R2 (see Table 1).

Cross-tree constraints.

There is information about dependencies or incompatibilities between features that is difficult or impossible to express in the hierarchical tree decomposition of features captured in the "Relationships" block. The "Constraints" block fulfills that need by allowing to write a wide spectrum of constraints related to basic and extended variability modeling (explained in the Subsection 5.2). This refers to requirement R2. Probably, the best-known examples of basic cross-tree constraints are "requires" (e.g., A requires B - the

```
1 someBetween 2 and 3 {
2     distractors
3     occultants
4     specialClimateConditions
5 }
```

Listing 2 Cardinality-based groups example

```
1 Attributes:
2 @NT string scene.comment
3 @RT int vehicle.speed [0..130] delta 5 default 40
4 real man.speed [0.0..30.0] delta 0.5 default 3.0
5 enum vehicle.identifier ["HummerH2","AMX30"]
6 real man.appearance_change [0.0 .. 1.0] delta 0.1 default 0.5
7 int *.cost [0 .. 1000] default 150
```

Listing 3 Feature attributes examples

selection of feature A in a video implies the selection of feature B) and "excludes" (A excludes B - feature A and feature B can not be part of the same video.).

5.2 Extended variability modeling

We now present more advanced mechanisms such as multi-features, attributes and default values.

Multi-features. VM employs cardinalities[5] before each feature name to specify the minimum and the maximum configurable copies it can have. To create a configuration of a video, a multi-feature and all its children features/attributes are cloned into copies, and each copy can be configured individually. This contributes to requirement R3.

The specification of a multi-feature follows one of two patterns placed before a feature name:
(1) `[`*minVal*`..`*maxVal*`]` (Listing 1, Line 12), or
(2) `cloneBetween` *minVal* `and` *maxVal* (Line 13).

In both patterns, *minVal* and *maxVal* are the minimum and maximum number of allowed feature copies. An example is given in Listing 1, page 10 (see Lines 12–13). There should be at least one vehicle in a video but the the feature vehicle can be cloned 5 times. The five instances of a vehicle can be configured individually with other attributes like speed or cost (see Listing 3). In our example, we can also have 10 instances of man. Overall, a video scene can have several vehicles and men, each having their own speed, appearance or cost.

Attributes. VM provides the "Attributes" block that defines properties associated with the features expressed in the Relationships block. VM supports basic types (boolean and not boolean) and enumeration attributes. Listing 3 shows 6 examples of attributes of types integer (int), enumeration (enum), float point (real) and chain of chars (string). This addresses requirements R3 and R6 (see Table 1).

Attribute domains. We provide support to bounded attributes that have a value between a range of numerical values (see requirements R3 and R6). The most basic bounded attribute is the one that has a fixed value. The value of those attributes cannot be changed in any configuration and are comparable to constant values in programming languages. For example: "real man.speed = 10.5" means that if a video has one or more men objects, their speed is always 10.5.

Some attributes of our running example are bounded. Line 2 stores a comment for each scene (@NT, @RT and deltas will be introduced in the next Section). Line 3 defines vehicle speed as an integer number that ranges between 0 to 130. Line 4 defines man speed as a real number that ranges between 0.0 to 30.0. Line 5 means that "vehicle.identifier" receives only one of two possible values, "Hummer" or "AMX30" (these are just two of the available vehicles models). Line 6 means that "man.appearance_change" receives a floating point value normalized between 0.0 to 1.0, and finally, Line 7 defines an attribute "cost" that ranges between 0 to 1000 and is assigned to all the features.

Default values. VM allows to the developers to establish a "default" value among a range of values and associate it to an attribute. For example, Line 3 defines that the vehicles' speed is 40, unless developers set other value in a configuration. In the MOTIV project, generation of video configurations is fully automated from the VM model. Therefore, default values are used during variables initialization using a CSP instance. In our current implementation we achieve this by defining a custom search strategy within the CSP solver that starts by evaluating the default values. Overall this construct addresses requirements R6 and R7.

[5] The cardinalities apply here to a feature, whereas a cardinality-based group (see previous section) applies to a feature group.

```
1 @name "scene variability"
2 @version 1.0
3 @description "Part of the variability model related to the scene"
4 @author "DiverSE Team"
5 @email benoit.baudry@inria.fr
6 @organization "INRIA, Rennes, France"
7 @date "March, 2014"
```

Listing 4 Model information block example

```
1 Descriptions:
2 feat man is "..."
3 att man.appearance_change is "..."
4 att *.cost is "the cost in miliseconds of adding a feature in a video sequence"
```

Listing 5 Descriptions block example

Model information, descriptions, and packages. VM provides the "Model information" and "Descriptions" blocks that allow developers to add meaning to VM models. Listing 4 shows an example of model information which includes tags such as "@version", "@author" and "@date".

The Descriptions block contains a list of definitions of features, attributes or constraints expressed in natural language (described in the next section). Listing 5 shows three examples describing the feature "man", its attribute "man.appearance_change", and the the attribute "cost" that applies to all the features.

This kind of information helps to document the rationale behind a feature or attribute, the justification of a default value or a constraint involving features and attributes. For reusability and maintainability reasons, the language also allows users to declare and import packages. For example, users can declare "package *packageA* { Relationships: ... *featureX* ... }", and "package *packageB*" enclosing attributes that reference the features defined in "packageA". Import declarations have a very standard form: *packageB* { import *packageA*.* ... att *featureX.attributeY* ... }". Packages and import declarations can be used for other types of blocks (e.g., for specifying specialized constraints). To summarize, this construct addresses requirement R4.

Finally, VM helps to improve reference integrity using well-formed rules: i) Only elements previously written in other blocks can be defined; ii) it is not possible to have two root features when importing Relationships; iii) attributes must indicate the feature where they are contained using the containment designator "."; and iv) attributes that apply to more than one feature could use the wildcard "*" instead of the name of the feature (e.g., cost in Line 4). These well-formed rules have been implemented and are controlled by the Xtext editor we have developed for supporting VM.

5.3 Scaling Up Variability Modeling

VM models do not act as contemplative documentation artefacts. The actual goal is to generate a certain number of *configurations* (corresponding to assignments of features values and attributes). In addition, the configurations should not violate the inherent constraints between features and attributes. More specifically, we aimed to generate configurations offering t-wise (e.g., pair-wise) coverage. T-wise criteria covering sets aim to provide good error detection while keeping low the number of tests to execute. This is usually achieved by granting that at least "t" input interactions of a program are tested. While is it possible to assign any discrete value to the "t", Cohen et al. [24] proved that covering the 2-wise criteria the 80% of errors (caused by feature interactions) are detected in a software system. For example, if our system takes three variables as input called A, B and, C; and we want to apply a 2-wise criteria we would need to cover the interactions between A and B; A and C and; B and C. Beyond t-wise criterion, other sampling strategies can be considered as well, both relying on solving techniques (e.g., CSP solvers).

We noticed that the video experts wanted to control and scale up the generation process (e.g., by fixing some values or stating that a feature should not be part of a configuration). We thus equipped and extended VM with "extra variability" modeling mechanisms to provide a scalable and purposeful reasoning over models. In the remainder, we describe the specific requirements we addressed and the corresponding solutions to support extra variability modeling.

Delta values. A solution for requirement R6 is a new construct called "delta". Each delta reduces the number of acceptable numeric values, therefore, "real man.appearance_change [0.0..1.0] delta 0.1" in the

```
1 urban requires vehicle
2 countryside requires clonesOf man < 10
3 countryside -> vehicle.dust == vehicle.size
```

Listing 6 Constraint block example

Listing 3, Line 6 will be interpreted as "enum man.appearance_change [0.0, 0.1, ... 0.9, 1.0]". This construct can be considered as a shortcut or syntactical sugar for defining a set of linear constraints. They also describe the precision required for properly picking some values.

Complex constraints. Our solution to the requirement R5 is to extend the basic "Constraints" block to allow writing a much wider spectrum of constraints that includes not only features but also operations and functions capable of handling attributes and features.

Listing 6 shows three examples of constraints, the first means that the selection of an urban background requires the selection of the feature vehicle. The second constraint means that the selection of a countryside background implies to include less than 10 different men along the video, and the last constraint specifies that the size of the dust cloud behind a vehicle in the countryside is equal the size of the vehicle.

Each constraint must be a valid expression that combines variables referencing features and attributes names, functions and operators. VM provides operators whose syntax, semantics, and precedence are very similar to the Java language. For example, VM supports arithmetic (e.g., $*, +$), relational comparative (e.g., $>=, <$), equality ($==, !=$), logical operators (e.g., requires, $\&\&, ||$) and conditional ($? :$) operators.

VM also allows to use functions as parts of the constraints. As an example, lets imagine that we want to restrict the time taken to generate a video to be less than 2 hours measured in milliseconds. That constraint is written as "sum (*.cost) < 2*3600000". Functions can accept several parameters or may have equivalent operators (e.g., sum and +). VM supports logical (e.g., xor, or, and), arithmetic (e.g., sum, avg, max, min) and sets (e.g., clonesOf) functions.

Multi-range and multi-delta values. Our solution to requirements R6 and R7 is to extend the definition of attributes with information about the several allowed ranges of values. Each range can have a different delta that reduces the number of values considered in each range (a.k.a. *multi-delta*). Multi-deltas are necessary when not all values of an attribute are equally important for creating configurations. The rationale is that in the delta should be small in ranges that are not too important, to consider just a few values in that range. For example, the attribute "luminance_dev" could have three ranges:
"real signal_quality.luminance_dev [0.0 .. 8.0] delta 1.0
[8.0 .. 32.0] delta 2.0 [32.0 .. 64.0] delta 4.0".
It allows users to decrease or increase the deltas on some subset of values that are considered as more relevant to discretize. In a sense, domain experts can prioritize the range of values that deserves more exploration and variability (requirement R7).

Run-time annotation. The solution for requirement R8 is to apply an annotation to indicate the binding time of each element. For the video domain we only consider run time but there are other binding times in other domains, e.g., load time, and link time. In our case, run-time features or attributes have values that can change in the course of a video sequence. A run-time feature or attribute is represented using the tag "@RT" or "@RunTime" before its name. (We also require that a run-time feature or attribute has a default value.) As an example, the "speed" attribute of a vehicle is considered as run-time since it can be changed during the course of a video. Line 3 of Listing 3 (page 11) shows that the speed of vehicles varies during the video between 0 and 130.

When a run-time tag is used, we consider that the responsibility of managing features or attributes' values is delegated to the Lua code. In particular, the Lua code can increase or decrease values at runtime (the handling of constraints at runtime is out of the scope of this article). The CSP solver does not seek to make vary the values of a "@RT" variable when sampling and generating configurations (i.e., the default value is used while the CSP solver controls that constraints of VM model hold). The benefit is that it simplifies the reasoning process since not all domain values are considered in the CSP problem. The solver can focus on diversifying other domain values when sampling configurations. Another interest is to document a feature or an attribute (see requirement R4).

Not translatable and not decision annotations. A solution for requirement R7 is the use of annotations that can be attached to features or attributes. This information states how to deal with their values when reasoning about a VM model with a solver. The *not translatable (NT)* means that a feature or attribute contains information that should not be considered when creating a CSP based on the VM model. It should be noted that NT variables should not be involved in a constraint (a well-formed rule verifies this condition).

```
1 objective generate_low_lumminosity_configurations {// night scenes
2  min (sum (*.luminance_mean)+ sum(*.luminance_dev))
3 }
```

Listing 7 Objective function example

A not translatable feature or attribute is represented by "@NT" or "@NotTranslatable" before its name. For example, in Line 2 of Listing 3 we considered the attribute "comment" of a scene as not translatable. This is especially useful for t-wise calculations using a CSP since it allows us to narrow the number of variables.

The *not decision (ND)* annotation means that a feature or attribute is a non-decision variable in a CSP problem. It is represented using the tag "@ND" or "@NotDecision" before its name. It should be noted that non-decision variables are part of the CSP problem and related constraints are still enforced. However the solver does not seek to proactively make vary their values (e.g., when sampling). This reduces the complexity of the CSP calculation because ND tagged elements will exist in the problem but the solver does not traverse all possible domain values during the search. In other words, values of ND tagged variables are computed by side-effects. We rely on the Choco solver to implement ND annotations as part of the CSP problem[6].

Objective functions. For further controlling the sampling of configurations (requirement R7), users need to specify features or attributes' values they desire. Objective functions force the selection of some values, and can be seen as an additional mechanism to restrict the configuration space. A solver can exploit objective functions as part of a CSP problem. Listing 7 shows an objective called "generate_low_luminosity_configurations" in the "Objectives" block. This objective is defined as the minimization of the total luminance, which is defined as the sum of the values of the attributes luminance_mean and luminance_dev. By minimizing the luminance, the video generator now synthesizes video sequences with night scenes. This objective function was discussed and implemented as part of the MOTIV project.

6 Evaluation

In this section we evaluate our variability-based approach, including the design of VM. We answer to the following research questions in different sections:

- *RQ1: What are the practical considerations of applying our variability language VM?* Section 6.1 relies on different dimensions of the framework of Savolainen *et al.* [27] for reporting on our variability experiences carried out in an industrial setting.
- *RQ2: How effective are VM language constructs for reasoning?* Section 6.2 evaluates the effectiveness of annotations for scaling up the generation of testing configurations.
- *RQ3: What are the practical benefits of a variability-based approach?* As we are in an *applied research* context, we aim to identify the improvements of our proposal with regards to existing industrial practice (see Section 6.3).
- *RQ4: What are the commonalities and differences between constructs of VM and state-of-the-art variability languages?* Section 6.4 discusses the literature and provides a comparison table.

6.1 Practical Considerations (RQ1)

We report on some practical considerations of applying VM, i.e., we address *RQ1*.

The goal is to help external readers or practitioners to evaluate if our particular experience and language are good for their purposes. Most of these considerations were proposed by Savolainen *et al.* [27] which are based on practical experiences and research carried out in cooperation with several companies – as also happened in our case[7].

Cost-Benefit. *"What is the optimal model in terms of cost-benefit when taking into account construction, usage, and maintenance?"* [27].

[6] See e.g., Section 2.1 of the Choco Solver documentation available here: `http://www.lirmm.fr/~lazaar/imagina/choco-doc.pdf` (August 2012)

[7] The classification of Savolainen *et al.* [27] helps to *"clarify the intent of a proposed method and [...] help practitioners in clarifying the guiding principles for their feature modeling"*. Therefore it is not a comparison framework; we use it as a means to properly report on our experience. A cross-validation with other empirical studies is an interesting research direction, but out of the scope of this article.

Construction is divided in different parts the creation of the language and the creation of the model of video variability: *Language infrastructure construction.* Creating a language, editor and interpreter is not for free since there are many tasks to support, such as parsing, auto-completion, syntax highlighting, etc. However, we discovered that new frameworks for language development make these tasks less complex and or even fully automated. For instance, we used Xtext[8] to generate the VM editor and parser, based only on the VM grammar definition. Using Xtext, we expended about one hour to create a working VM editor for the first iteration of VM, three days for the second iteration, and about four days for the third iteration. Admittedly, the most difficult part was to understand how to model and interpret nested expressions and operators precedence in the constraints block.

An extra effort for us was the connection with the Lua code and the configuration files for the VM model to be aligned with the schema of the configuration file exploited by the video generator. We needed technical exchanges (by emails), beyond meetings with video. This part took about one week, as it was a common effort between the creators of the language infrastructure and the creators of the video generator.

Model creation. This task was the one that took more time; it required to understand the domain, the requirements, and to discuss with video experts. We produced six different versions of the video variability model during a period of about nine months. These versions were made after four large meetings with all the project partners (these meetings focused on different topics apart from variability modeling, including administrative issues and technical issues in the video analysis domain), and two individual meetings with the main developer of the video generator.

Usage. Just a sketch of a feature model would be enough for communication; however, our project justified the construction of a language and tool to support not only communication but also enable video generation through scalable automated analysis.

A manual writing of a valid Lua configuration file takes around three minutes for the video generator expert. Manually creating an initial set of only 500 videos would thus take hours. There is also the risk to set invalid values and to create invalid configurations. VM supports the process to generate not only 500 but also thousands of valid and diverse video configurations that guarantee some objectives (e.g., pair-wise coverage) in seconds. Section 6.2 complements the cost-benefit point with an evaluation of the benefits in terms of performance scalability of the new constructs proposed by VM.

Maintenance. In our particular experience, we did not experience significant maintenance costs associated with changes to the language grammar or the video variability model written in VM. On one side, Xtext helped to us maintain the language infrastructure code (e.g., parser, editor, etc.) by separating generated code from manual code. On the other side, we did not experienced major problems to update our video variability model since the video domain is stable and the only changes that we applied were increments in the specification.

As a conclusion for the cost-benefit practical consideration, we can say that the costs of constructing, using and maintaining VM models are low compared with the benefits of producing automatically suitable videos to test complex video analyzers. Similar achievements were impossible before the introduction of variability techniques.

Completeness. *"How complete is the feature model?"* [27].

In our industrial settings, the resources were limited and, in general, there is potentially an infinite number of possible videos to consider. Therefore we only selected characteristics and physical properties of videos we were capable to realize with computer vision procedures and Lua code. The scoping process was also driven by the diversity of situations we wanted to cover. Overall, the VM model expressed a comprehensive set of possible videos, capable of testing algorithms in numerous contexts.

Besides we have decided to not include in the VM model some parameters of the Lua code. In particular, we did not model the time and order in which events happen in a video sequence or the path of moving vehicles and people in an scene. It was judged too complex to express in a variability model and beyond the scope of the VM language. The number of valid configurations in the final VM was already considered as high enough.

Stakeholders. *"Who puts effort into and who gains the benefits of the model? What knowledge about feature modeling methods in general and the product line in question do the stakeholders have?"* [27].

VM was developed mainly by a team composed of two people (a doctoral and postdoctoral researcher) and one lecturer at Inria, which knew about product lines and feature modeling methods. This team created the language infrastructure, implemented a translation from VM to a CSP (presented on a previous work [10]), and worked to connect the VM tool with the video generator.

The main developer of the Lua video generator is a video expert that provided feedback for improving the VM design. In addition, he wrote an initial description of the important aspects that may be varied

[8] http://www.eclipse.org/Xtext/

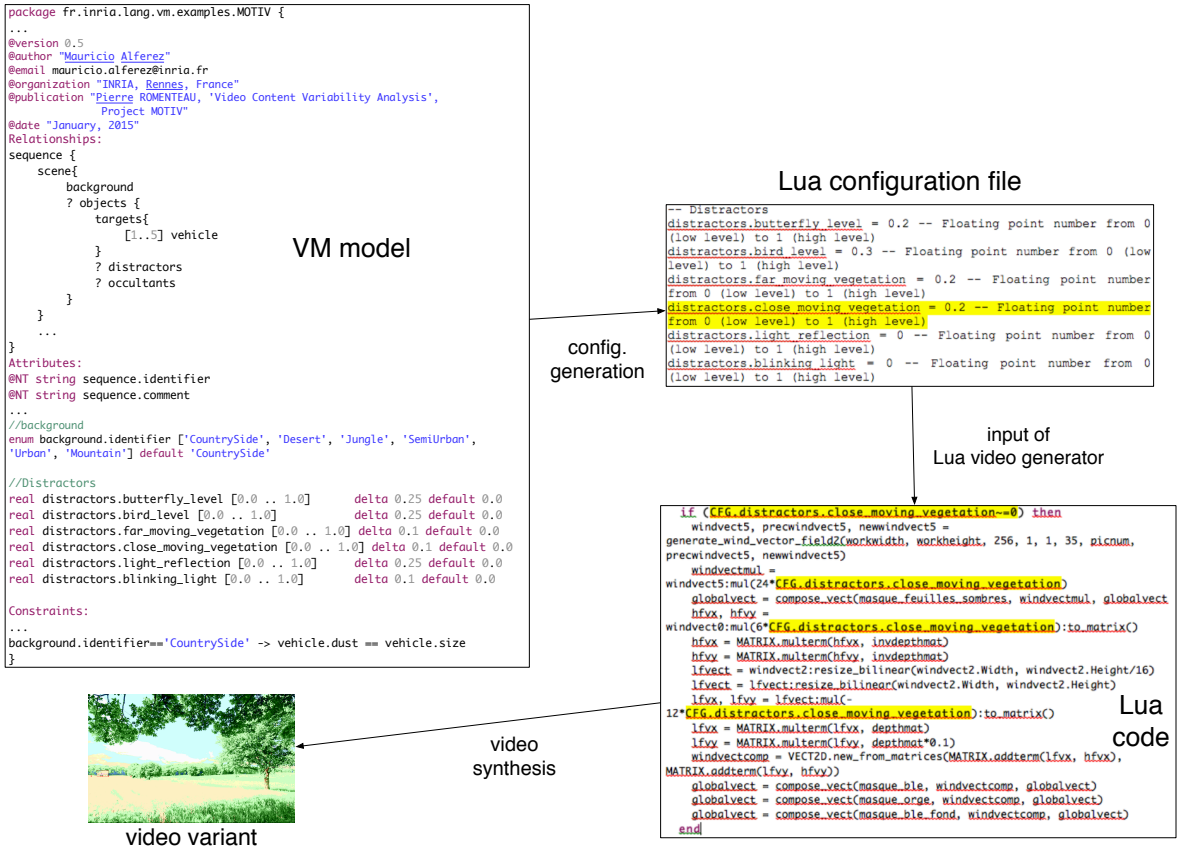

Figure 4 From a VM model, we generate configurations that are fed to a Lua generator for synthesizing video sequences. VM configurations are obtained through sampling and constraint solving (see previous sections). A VM configuration is translated into a Lua configuration file (each feature/attribute has a corresponding Lua parameter).

in a video. Based on the description, the development team wrote the first version of the VM model and used that version to communicate with the rest of the partners in the following meetings. Stakeholders from the DGA provided comments that were addressed in the following iterations and model versions. The main developer of the video generator also controls that the VM model follows the evolution of the parameters in the Lua code (see Figure 4).

Taking into account the variety of stakeholders, we took the decision of dividing the VM language by blocks, each one addressing a different concern. Video experts without too much technical expertise can focus on concerns described in the relationships, model information, definitions, and objectives blocks. Developers and video experts with a programming background can focus on adding annotations, constraints, deltas, or further defining the objectives and attributes blocks.

It should be noted that we showed and shared the VM model to stakeholders, but they rarely used the VM language for directly editing the models. They rather provided suggestions and feedbacks through natural languages (e.g., emails) and we take the responsibility of integrating the changes into VM models. Overall, we found that the basic feature modeling concepts were well understood in such a way stakeholders can participate in the evolution of the VM model. However, we have not evaluated the ability of stakeholders to directly use VM; it would require controlled experiments that are out of the scope of this article.

Domain. *"Does the model represent the problem or solution domain? Does the model represent a current or planned product line?"* [27].

VM can be used to model either the problem or solution domain; both sides influence the design of VM or to represent a current or future product line. In our experience, we first modeled the video domain from a problem domain perspective (during meetings with domain experts), and then realize variability through configuration files and Lua code.

Commonality. *"How much commonality is represented?"* [27].

Although VM mainly targets variability modeling, it also supports commonality modeling through the notion of mandatory features. Attribute values can be fixed as well. Such language mechanisms for specifying

commonality emerged as a means for specializing the variability model to specific testing scenarios. For example, we can fix the value of the luminance_mean attribute in order to force the generation of a certain kind of videos.

Correspondence. *"What elements of the product line does the feature model correspond to?"* [27].

Mapping. Many features in the video domain VM model have a 1-to-1 relationship with code modules that implemented the video generator. In a similar way, feature attributes tend to match input parameters of Lua functions. 1-to-1 mappings between features in the problem space (VM model) and their realizations in the solution space (Lua code) eases their co-evolution. In Figure 4, we give an excerpt of a VM model, a possible configuration file, and the Lua code. We highlight in yellow the portion of Lua code related to close_moving_vegetation.

Abstract features. Using 1-to-1 mappings is not a strict rule. In fact, we also modeled features that are not mapped to any specific module to group other features or attributes. For example, the feature "objects" does not map directly to any module, but helped to group conceptually the "vehicles" and "man" features that have concrete mappings to the code.

Constraints. *"What do the constraints represent?"* [27]

VM addresses the challenge of managing and representing constraints through a set of functions and operations over features, attributes and sets of features (e.g., "ClonesOf"). Constraints are also important for *specializing* the VM model to specific testing scenarios. For instance, experts want to synthesize only videos with a specific background (such as desert or urban) or luminance; some values are thus fixed, but the other features or attributes are still subject to variations.

Notation. *"What constructs and representation should different stakeholders use?"* [27]

The VM language provides a *textual* notation for expressing variability. There are two major reasons. First, some of the partners are developers of video algorithms and are already familiar with textual content. In contrast to diagrammatic languages, participants continue to use well-established efficient tools in the industry such as code editors. Second, numerous attributes, meta-information, and cross-tree constraints have been specified; by construction they are textual information.

6.2 Evaluating the Impact of New VM Constructs for the Reasoning (RQ2)

We now evaluate a major automated analysis operation for sampling configurations based on the computation of a pair-wise coverage [28,10]. The operation takes as input a VM model and generates some configurations (i.e., values for features and attributes) conforming to the constraints. Our goal is to study the effect of (1) @ND (for "not decidable") and (2) deltas (for varying the increment of a domain) on the performance of the operation. We expect to decrease the amount of time using meta-information (@ND and deltas).

Data. For the two experiments, we took the complete VM model of the MOTIV project as input. This model contains: i) 18 features containing different amount of attributes; and ii) a total of 84 attributes with ranges going from 0 to 120000. The size of the sum of all ranges represents 2161711 integer values. We estimate this model represents up to 10^{100} configurations. A key characteristic of this model is that most of the variability is represented as numeric attributes related to physical properties.

Experimental settings. The two experiments were executed in a Dell computer running an Intel i7 M 620 at 2.67GHz and 4 GBs of RAM. The operating system was Ubuntu 12.04, with a 1.7 open-JDK virtual machine. The implementation of the pair-wise operation internally relies on the Choco 2 solver.

Evaluating the effects of @ND. In the first experiment, we created ten groups, each one containing ten copies of the original model. Each group has a percentage of @ND tags, which went from 0 to 100 percent. The tags in each group were assigned to the attributes randomly. We report on the average time required by each group. The experiment hypothesis is that the use of @ND tags improves the performance of the pair-wise operation in the context of the MOTIV project.

Figure 5 shows the results. The time varies around 10 seconds between the models containing 100% of @ND tags and the models with no tags. It represents an improvement of around 30% in the execution time. The improvement is significant. But the testing operation can still scale in a reasonable amount of time without @ND tags. At this step of the research, we thus cannot state that @ND tags are mandatory constructs in MOTIV for scaling but it matters and helps improve the time needed. This is, we suspect that it would be more useful for larger models; this experimentation is kept as future work.

Yet, we conjuncture that the VM model of the MOTIV project will grow in complexity and handle more attributes and features in the future. We expect to gain even more time reduction in future releases based on the Figure 5 tendency. Another argument for @ND tags is that we generate only *relevant* configurations.

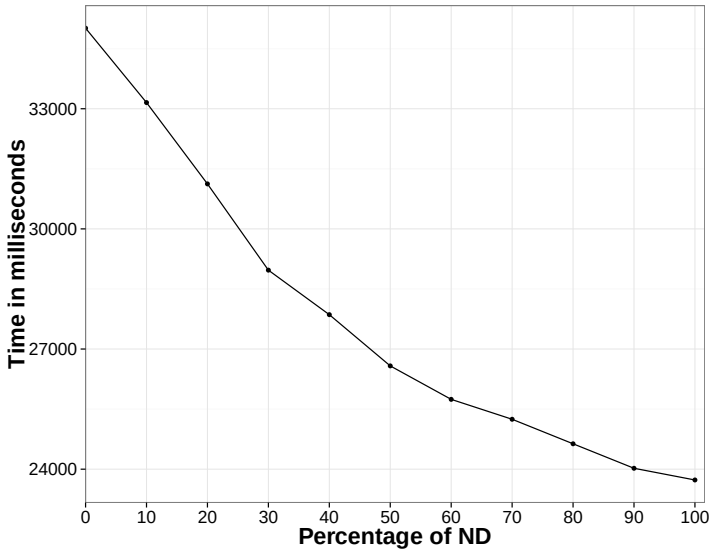

Figure 5 Time for obtaining a pair-wise coverage depending on the percentage of ND for the MOTIV feature model.

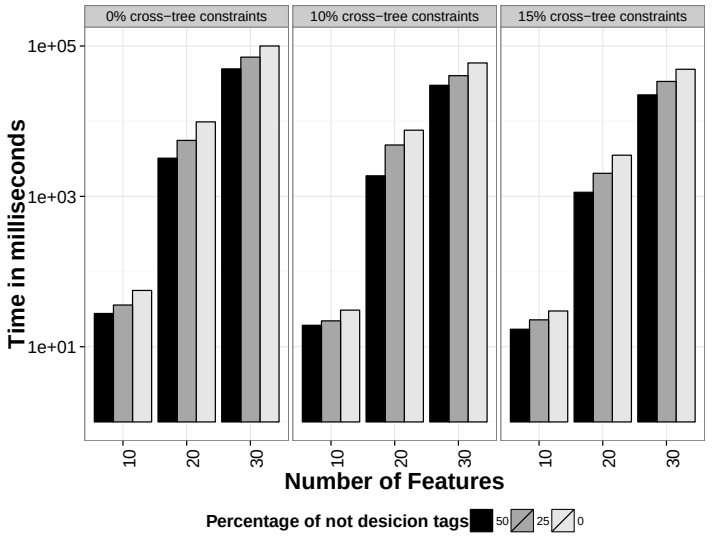

Figure 6 Time for obtaining a pair-wise coverage depending on the percentage of constraints and ND (random feature models).

Note that ND tags helps the solver to focus only in decision variables but this does not reduce the complexity of the problem in terms of variable domains.

Scalability improvements in random models. The major bias of our previous experiment is the population validity. Therefore, we extend our experiments to variability models having different feature hierarchies and attributes nature. We performed the same operation over a set of models generated by Betty tool [29]. Betty is a generator of random feature models with different parameters to control the number of features and attributes, percentage of cross-tree constraints, etc.

Figure 6 shows the time required by the operation depending in the amount of cross-tree constraints and the percentage of non decision tagged attributes. It is remarkable that the time required by the operation is reduced almost in the same percentage as the annotations introduced. Moreover, when models are big enough (e.g., 100 features) the time reduction is more than 30 minutes. This points out that the introduction of extra information is handy for providing better results when implementing automated analysis tools.

Evaluating the effect of deltas. In the second experiment, we measured the impact of deltas usage in the pair-wise operation. Specifically, we compared the time required to execute the operation with and without deltas. When no deltas are specified, we consider an increment of "1" for integer ranges. We executed

the testing operation with the deltas provided by our industrial partners. In this experiment no tags were used so the improvement of each language construct can be evaluated independently.

We first observed that the variable domains were reduced in 3400 integer units in the CSP when using the deltas optimization. The pair-wise operation without deltas took 38 seconds. When enabling the deltas usage, the solver took 34.8 seconds. This represents an improvement of 3.2 seconds. This experiment shows that the constructs of the language improves the scalability.

Discussion. The improvement is noticeable but does not impose the presence of deltas at this step of the research. As for @ND tags, the potential of deltas may be more apparent with the growing complexity of the VM model. Overall we envision the combined use of deltas and @ND to reduce the amount of time and generation of relevant testable configurations.

Another interest of delta lies in the *control* of the configuration generation: Practitioners can fine-tune the way values of attributes vary. For example, for the same range of possible real values 0..1, some attributes vary by 0.1 whereas others vary by 0.25 (see `https://github.com/ViViD-DiverSE/VM-Source/` for the VM models). The discretization of possible values differs depending on the precision needed. Discussions with video experts revealed that the specification of deltas is specific to each attribute; they should control deltas such that our reasoning operation can compute relevant and diverse configurations.

6.3 Discussion: Benefits of variability (RQ3)

In this section, we retrospectively compare the three approaches used throughout the project. In *applied research*, the objective is to create technology that is better in some manner than those already developed [25]. We aim to address *RQ3: What are the practical benefits of a variability-based approach?* Specifically, we want to show the improvements of a variability-based approach in terms of *automation* and *control* thus participating to the creation of *large-scale* datasets (videos).

Non generative approach (A_0)

At the starting point of the project, the testing practice was either to collect existing videos or to film new videos. Then algorithms perform over the videos and metrics are computed to determine the accuracy or the response time. Based on the results and statistical methods, practitioners can determine the strengths and weaknesses of their solutions.

We recall here two severe limitations. First, not only the collection of videos is a costly and time-consuming activity but also the annotations to specify what are the expected results and thus evaluate the algorithms. Another limitation is that the collected dataset is usually small in size and not representative of testing scenarios. Some benchmarks exist (for example, for event recognition, see, e.g., [20]) but are specific to vision analysis tasks and cannot be seamlessly reused (e.g., for military scenarios).

In summary, the practice we have observed at the beginning of the project suffers from a lack of automation – precluding the establishment of large datasets – and a lack of control over the testing videos.

With the generator (A_1)

With the development of a video generator (see Figure 1, page 4), practitioners can envision to build a larger, more diverse, and representative dataset of videos for testing their algorithms. At that time, the elaboration of a dataset consists in setting some values to a configuration file and then executes the generator to produce a variant.

Compared to a non generative approach (A_0), the use of a video generator has the advantage of providing (1) more automation: there is nor the need to film neither to annotate videos; (2) more control: practitioners can tune the parameters to produce the video variant they want.

However some limitations remain. The approach still requires an human intervention for specifying *each* configuration file. The setting of values is tedious and impractical when a large number of configuration files has to be set. This is evident in the current VM model that notably describes 84 attributes (out of which a large proportion are reals) and 2161711 individual[9] attribute values in total can be set.

With a manual elaboration of configuration files, practitioners eventually ignore what test cases (video variants) are covered. Moreover it is hard to augment the dataset because of the lack of automation and

[9] Each attribute has a domain size, which corresponds to the number of individual values an attribute can take. We consider *deltas* (see Section 5.3, page 12) for the computation of domain size. It should be noted that the number of possible configurations is significantly greater than the sum of possible individual values – since a configuration is a combination of individual attribute values.

the lack of knowledge of what test cases are missing. This covering knowledge is very important since the most situations are covered, the more practitioners are confident in terms of robustness, performance and reliability of their algorithms. It is especially important for an institution like DGA to have a strong coverage guarantee. It is as important for the two industrial partners to cover a maximum kind of situations and determine if the algorithms behave accordingly.

Another limitation related to the previous observations is the difficulty of controlling the synthesis for the synthesis of *specific* datasets. For instance, the synthesis of video variants in which the global luminance only varies between, say, 0.6 and 0.8, is tedious and error-prone. In this case, practitioners have to manually set the value while ensuring it is not dependent of another parameter; the random modification of the luminance values and the whole process should be repeated for each configuration file. It is again impractical w.r.t. to the number of attributes and possible domain values.

In summary, the development of a video generator still suffers from a lack of automation – precluding the establishment of large and diverse datasets – and a lack of control over the testing videos.

Variability-based approach (A_2)

The use of a VM model to pilot the generator allows practitioners to have more automation and more control.

Instead of manually modifying each configuration file (see A_1), an automated operation processes a VM model and fully generates all configuration files. The effort of the practitioners is thus dramatically reduced. Another benefit is that constraints over or across parameters' values are valid by construction.

A variability-based approach also helps to *specialize* the synthesis. Different alternatives can be employed for this purpose:

- putting additional constraints and specializing the VM model for specific scenarios. For instance, a specific Background (e.g., Urban) can be set up since the application is known to be deployed in a specific military ground. In turn the testing machinery will then consider only configurations with Urban. The benefit is that practitioners can focus on specific testing scenarios, specializing the test suite to realistic cases;
- optimizing different objective functions over attributes: practitioners can specify the relative importance or cost of a feature, fix some parameters, etc. Again it aims at customizing test suite to fit realistic needs;
- precluding some features or attributes, not relevant for testing, with the use of meta-information.

In terms of *covering*, a variability-based approach grants, by construction, the validity of the T-wise (e.g., pair-wise) criterion. The covering criterion can be combined with other specialization mechanisms.

Realism and exploitation of video variants. We have conducted three initiatives to validate some of the video variants. First, at the end of the project, experts have reviewed 60+ video variants we have synthesized with the generator. They judged that the video sequences were *visually* realistic and can be used to test some algorithms.

Second, experiments made by the partners showed that some house-made video algorithms have difficulties when processing certain kinds of videos. We used this time a larger sample of 500 videos. Specifically, the precision and recall of some algorithms were not satisfactory for some video variants. Such results are promising since they show that video variants can be used as test cases to identify weaknesses of video algorithms. Our long term goal is to further investigate the use of synthetic video variants for testing vision algorithms.

Third, our recent development [16] on top of VM further enforced the *quality* of the video generator. We developed an automated procedure capable of detecting videos that are not of interest for computer vision algorithms and humans. Typically, these are videos in which the vision system cannot perceive anything or cannot distinguish moving objects from other ones; we used a sample of 4000 video variants (more details are given in [16]). Based on machine learning techniques, we were able to automatically inject constraints into the variability model for avoiding the generation of irrelevant videos. This increase in quality was possible because of an explicit variability model. Without such an abstraction and without a variability approach in general, we simply could not realize our idea and thus enforce the video generator.

In summary, the introduction of variability techniques on top of the generator induces important benefits in terms of automation and control. Practitioners can now synthesize large, suitable, and diverse datasets – something practically impossible with previous practices A_0 and A_1.

Characteristic / Approach	FODA [38]	FDL [39]	SXFM [40]	FAMILIAR [22]	VELVET [41]	UTFM [42]	CLAFER [23]	Saloon [43]	VSL [44]	TVL [45]	Gears	Pure::Variants	FAMA [21]	VM
Multifeatures	○	○	○	○	○	●	●	●	●	●	○	○	○	●
Attributes	○	○	○	○	○	●	●	●	●	●	●	●	●	●
Default values	○	◑	○	○	○	○	●	○	●	●	●	●	●	●
Deltas	○	○	○	○	○	○	○	○	○	○	○	○	○	●
Elements definition	○	○	●	○	○	○	○	○	○	○	●	●	●	●
Constraints	◑	◑	◑	◑	◑	●	●	●	◑	●	●	●	◑	●
Multi-ranges and multi-deltas	○	○	○	○	○	○	○	○	○	○	○	○	◑	●
Run-time anotation	○	○	○	○	○	○	○	○	○	○	○	○	○	●
NT	○	○	○	○	○	○	○	○	○	○	○	○	○	●
ND	○	○	○	○	○	○	○	○	○	○	○	○	○	●
Objectives	○	○	○	○	○	○	●	○	○	◑	○	○	○	●

●addressed as goal, ◑addressed but with restrictions, ○not regarded as goal

Table 2 Summary of comparison between languages w.r.t. specific requirements of the MOTIV project and video domain

6.4 Comparison with Existing Variability Languages (RQ4)

Numerous languages, being textual or graphical, have been designed to model variability. For instance, feature models have become more and more sophisticated since 1990 and their dialects have been detailed in comprehensive surveys, for example, by Schobbens *et al.* [30], Benavides *et al.* [9], Galster *et al.* [31], and Eichelberger and Schmid [32,33]. A research question we now address is *RQ4: What are the commonalities and differences between constructs of VM and state-of-the-art variability languages?*

We have relied on recent comprehensive surveys [6, 32, 33, 31] to select the variability languages. We have also considered variability languages used in two major industrial tools pure::variants and Gears [34–37]. Table 2 summarizes the comparison of VM with some representative languages in terms of the characteristics they address as a goal.

Most common characteristics. Boolean constructs of feature models (as supported by FODA [38], FDL [39], SXFM [40], VELVET [41] or FAMILIAR [22]) are useful in the video domain, but not sufficient. New dialects (e.g., UTFM [42], CLAFER [23, 46], SALOON [43], VSL [44, 47], TVL [45] and FAMA [21]) have emerged to overcome the expressiveness limitations of feature models, for instance, to deal with *attributes* or *multi-features*.

Most of the languages do not allow to explicitly change a *default value* for features and attributes so, we considered that they do not address that characteristic as a goal. FDL is an example of a language that addresses this characteristic with restrictions. It includes the construct "default" however, it only uses it to declare a selected-by-default atomic feature in a group and not a default attribute value.

Another case of characteristics that are addressed only partially is the *constraints*. While constraints have been addressed by all the languages, in most of the cases they did not consider constraints including features, attributes values (e.g., in FODA, FDL SXFM) and multifeatures (e.g., VSL and FAMA).

Less common characteristics. The *main differences between VM and the other approaches* are mainly the use of meta-information associated with features or attributes. For example, VM users can include: i) *deltas*, ii) *elements definitions* –model, features and attributes information, iii) objective functions, iv) multi-ranges and multi-deltas, v) meta-information annotations such as "not translatable", "not decidable", and "runtime" for controlling the solver-based reasoning. As reported in Section 5.3, our industrial experience strongly motivates the introduction of these new constructs. We also show the importance of the constructs in terms of reasoning scalability (see Section 5.3).

Summary (RQ4). Table 2 shows that most of the individual constructs of VM can be found in some of the languages (or are addressed but with restrictions). However, all constructs cannot be found in a single and integrated language. Hence, VM can be seen as a unique combination of language constructs, tailored to specific industrial needs. There are also unique language constructs like deltas, multi-ranges, and meta-information annotations to control reasoning that are not explicitly addressed by existing languages.

Discussion: Is there a one-size-fits-all variability language? The comparison highlights two important aspects of variability languages. First, some *common* needs for modeling variability are emerging such as the support for attributes and multi-features. Second, *specific* constructs are also needed and were a prerequisite for successful adoption in the case of VM– similar observations have been reported in other domains (see Section 8 and in particular the discussion of references [6,48,14]).

Existing variability languages do not address some of the requirements of the MOTIV project simply because they have not been designed to. On the other hand, some languages for variability offer specific constructs that VM does not. For instance, VM does not support visibility conditions [48] or the advanced specialization mechanisms [23] of CLAFER, simply because we did not need them in our case. Therefore, we cannot claim that VM represents a one-size-fits-all solution applicable to any domain or software systems.

Our experience rather shows that variability languages need to be tailored for addressing specific requirements. It calls to further investigate mechanisms that would support the customization or extension of variability languages.

Discussion of [13]: Variability languages and empirical insights. Sepulveda *et al.* performed a systematic literature review of requirements modelling languages for software product lines [13]. The study includes variability modeling languages developed from 2000 to 2013.

Interestingly our work confirms some findings of the IST article [13]. First Sepulveda *et al.* report that *"some constructs (feature, mandatory, optional, alternative, exclude and require) are present in all the languages, while others (cardinality, attribute, constraint and label) are less common"*. Second there is a concern for generating proposals with higher levels of expressiveness. It is in line with our previous comparison of variability languages. Meanwhile our work contributes to the lack of empirical validation and adoption in industry (as identified in [13]). For example, it is stated that *"57% of the languages have been proposed by the academia, while 43% have been the result of a joint effort between academia and industry"*. Our research is precisely a tight collaboration with industrial partners to capture the right level of expressiveness for VM and to fully develop a video generator.

7 Threats to Validity

External Validity. There are two major external threats related to the scalability performance evaluation (see Section 6.2). Concretely, *(i) population validity*, i.e., the model used in the experiments represents only one concrete instance of the problem. We consider that the feature model is realistic since several experts were involved in its design. Moreover, the result is not a contemplative model and has proved to be effective to synthesize videos variants. It is also possible that the future evolution of the model changes its inherent complexity and influences the results. Another threat is that the variability model does not reflect properly the same complexity as other realistic models. In terms of modeling elements, it is a fairly small variability model with 18 features and 84 attributes. In terms of configuration complexity, numerical attributes lead to a combinatorial explosion of possible configurations, mainly due to real values. We estimate that the number of configurations is 10^{100}. That is, the number of configurations is very important despite the relative low number of features and attributes.

To mitigate this threat, we executed a second experiment considering random models. In this experiment, we used the Betty [29] generator that aims at mimicking attributed feature models. It is possible that they do not cover all the properties of real models. Also, we only measured this improvement using one feature model analysis operation from the thirty proposed in the literature [9]. *(ii) ecological validity*, while the experiments have been executed for maximizing the isolation of external threads in the machine, it is possible that third-party threads (e.g., operating systems threads) were jamming the results. To minimize the error introduced by them in our results, we executed the analyses 10-times reporting on averages.

The validation of video variants by experts has been realized on a sample of only 60 configurations. The reviewing of video variants is a time-consuming activity and yet is necessary for assessing the visual aspects of a video. To mitigate this threat, we have conducted two additional experiments (see end of Section 6.3) on a larger sample thanks to algorithms and automated procedures.

Construct validity. The scalability results are promising in terms of time required to solve problems related to our feature model. However, we might need to perform a higher scale experimentation when referring to multi-objectives configuration problems.

The construction of the comparison table (see Section 6.4) and three assessment values (addressed as a goal, addressed but with restrictions, not regarded as a goal) was useful to give an overview of common and less common characteristics and therefore, it addressed the research question. However, we might need to employ a more specific study to compare not only the characteristics of the languages but also the different

tools that support the languages (probably more than one tool by language) and therefore, we will need to include more criterion.

Internal validity. Another threat is that we evaluate the practical considerations (see Section 6.1) while being active participants of the project. To mitigate this threat, we structure the criteria according to an external evaluation framework [27]. Generalization of the observations of Section 6.1 (e.g., for the VM language or for the variability methodology) would require additional case studies and is premature at this stage of the research. The goal of Section 6.1 is thus more modest; we want to report on our specific industrial experience in a structured and disciplined way.

8 Related Works

Previous Section 6.4 specifically compares state-of-the-art variability languages w.r.t. VM and specific requirements we identified in an industrial project. In this section, we consider other existing works related to variability notations, domain-specific languages, reasoning support, test generation, and variability in the video domain. First, we review empirical studies that have considered variability notations used in open-source projects or in industry; it aims to complement the comparison of Section 6.4. Second, we present works related to domain-specific profiles or languages since VM can be seen as a specific language for modeling variability. Third, we discuss approaches for reasoning about feature models, an important requirement of the MOTIV project. Fourth, we establish connections with works in test generation since our approach aims to synthesize a set of video variants that can be seen as a test suite. Finally, we discuss works using variability techniques in the video domain.

8.1 Variability in the wild and in industry

Berger *et al.* [48] studied the modeling of variability in the operating system domain (Linux, eCos, and FreeBSD are the subjects of the study). They showed that well-researched concepts of FODA feature models, comprising Boolean (optional) features, a hierarchy, group and cross-tree constraints, are used. They also identified domain-specific concepts beyond FODA feature models, such as: visibility conditions, derived features, derived defaults, and binding modes.

Dumitrescu *et al.* [49,50] reported on their experience in an automotive model based systems engineering. Mussbacher *et al.* [51] propose an extension of the Aspect-oriented User Requirements Notation (AoURN) to support variability modeling. The outcome is a holistic reasoning framework based on goal modeling, feature modeling, and specification of scenarios. The framework has been applied on Via Verde, a real-world product family that aims to simplify the payment processes.

A recent survey reported that feature modeling is by far the most popular notation used in industry [6]. However no details are given on the specific language constructs used for modeling variability requirements. The industrial survey shows that pure::variants and GEARS are the most industrial tools used to model variability. They provide support for feature models but some adaptations are needed to cover all the requirements we faced in our industrial project. The most important was to provide reasoning support for extra variability and a mechanism to discretize multiple ranges of values defined in continuous domains.

Interestingly, a variety of notations is used in industry – most of industrial practitioners rely on several notations [6]. Studies of variability also show that modeling languages in open source systems all contained domain-specific, or even project-specific language constructs [48]. Nadi et al. [14] applied the variability language Clafer in the cryptography domain and suggests some language extensions for improving their models. Our experience-report also highlights specific needs when modeling variability in the video domain. It questions the existence of a one-size-fits-all solution applicable in any industry without specific adaptations.

The Common Variability Language (CVL) (`http://www.omgwiki.org/variability/doku.php`), a recent proposal for OMG's standard, describes a comprehensive process for modeling software product lines. CVL includes the description of a variability abstraction model (VAM) that conceptually corresponds to a feature model with attributes and multi-features. The language VM is compatible with the VAM of CVL, but also comes with specific constructs (e.g., meta-information) and an associated reasoning support.

8.2 Domain-specific profiles and languages

One solution to address specific needs when modeling variability is the use of modeling profiles. These are particular ways to give a host language the feel of a domain-specific language. For example, Hofman *et al.* [52] extended UML Activities to represent and relate different kinds of "Features".

Another alternative is the development of domain-specific languages (DSLs) [53–55]. We want to highlight the fact that VM provides no specific construct to the domain of video (e.g., the language construct "scene" or the keyword "illumination"). Therefore, we can consider VM as a domain-specific language for variability modeling in general; VM has proved to provide adequate variability constructs *for* the video domain and the industrial setting we have investigated.

We have collected some evidence of the applicability of VM as a generic attributed feature modeling language (such as SXFM, TVL or FAMA). First, we have implemented a transformation between FAMA and a subset of VM so that the two languages are now interoperable. Second, we have elaborated VM models to encode existing boolean and attributed feature models found in the literature and public repositories (e.g., mobile media [56] and the SPLOT repository `www.splot-research.org`). More details can be found in the repository of VM models: `https://github.com/ViViD-DiverSE/VM-Source`.

8.3 Variability and reasoning support

Benavides *et al.* [9] made a survey of more than 20 years of automated analysis of feature models. Most of the reasoning operations apply on FODA feature models, i.e., with Boolean constructs. It called for more research devoted to the formalization, performance comparison and support of so-called extended feature models. Since then, advances have been made to support attributes and multi-features (also called *clone enabled features*), relying on either CSP solvers, BDD, SAT, or SMT solvers (e.g., see [57–59,8]). An original and crucial aspect of our work is that we exploit meta-information over features and attributes when encoding VM models and generating test configurations. It has two merits: i) reducing the complexity of the constraint problem fed to the CSP solver, and ii) generating test configurations that contain only features and attributes *relevant* for specific video analysis scenarios.

In [10], we developed testing analysis operations operating over VM models (i.e., attributed feature models). Our previous work [10] focused on the testing operation. In this paper we comprehensively (1) describe the variability language and (2) report on our industrial experience. The effect of deltas and @ND ("not decision", see Section 6.2) on scalability performance had not been evaluated either in [10].

8.4 Test generation

There exists many previous works for automatic test generation [60,61]. Constraint solving techniques are intensively used for this purpose. We rely on the same foundations and VM can be seen as a specific solution for expressing and reasoning about constraints. Some of the existing works specifically focus on the generation and selection of test configurations for configurable systems [62–65]. These works focus on managing the combinatorial explosion of test configurations and on minimizing the number of configurations under test. However, they do not handle the generation of concrete data for testing these configurations, nor do they handle the oracle. In particular, previous work do not generate variants of video sequences together with ground truths, for video analysis software systems.

The variability model that we consider in this work captures variations among input data for the program under test. Hence, our reasoning technique for sampling the set of all variants can be considered as a special form of input space partitioning [66], in which we consider each t-wise combination of features as an equivalence class with respect to the program's behavior. The language VM provides advanced constructs to control in a fine-grained way the boundaries and partitioning of the configuration space.

8.5 Variability and video domain

There is a plethora of work related to the domain of computer vision (and by extension to video analysis). Many video algorithms have been designed and benchmarked, and form the basis of many crucial applications of modern society. An original goal of the industrial project is to synthesize variants of videos with the intention of testing video algorithms. To the best of our knowledge, no generative approach, guided by a

high-level variability specification and supported by automated techniques, has been proposed or developed in this domain.

Moisan *et al.* [67] and Acher *et al.* [68] proposed support to model the technical variability of video algorithms. The objective was to systematize the deployment of a customized video surveillance processing chain, suited to specific tasks (e.g., tracking of persons in an airport) and reconfigurable at runtime [67, 68]. In our industrial project, the goal and requirements are radically different: the challenge is to model the variability of videos – not of the algorithms. This key difference leaded us to design and use advanced variability language constructswhile Acher *et al.* used only Boolean constructs for modeling variability [68].

9 Conclusions & Lessons Learned

In an industrial project, we faced the original challenge of synthesizing video variants. The goal is to test competing vision algorithms and thus determine what solutions are likely to fail or excel in specific settings. It is crucial for the partners of the project – being providers or consumers of the algorithms – to collect a comprehensive and suitable input of videos. The current practice, based on the manual elaboration of videos, is very costly in resources and cannot cover the diversity of targeted video analysis scenarios. In this paper we introduced a generative approach and we addressed the following problem: What are the variability requirements in the video domain? How to capture what can vary within a video and then automate the synthesis of variants?

This paper reported how specific requirements, encountered in the project and in the video domain, have shaped the design of a textual variability language (VM) with advanced constructs and reasoning support. We learned the important lessons from our industrial experience. First, basic variability mechanisms *à la* FODA – Boolean (optional) features, hierarchy, group and cross-tree constraints – are useful but not enough and attributes and multi-features are of prior importance. Second, meta-information is relevant for (1) performing efficient computer-aided analysis of VM models, and (2) controlling the generation of testable configurations (e.g., to focus on specific attributes of features). Third, different iterations were needed to identify and implement additional specific constructs (e.g., deltas and binding mode) when connecting VM to the video generator developed by the industrial partners [11]. Finally, we learned that variability modeling can be effective in complex domains involving highly configurable systems. We are now able to synthesize 300Gb of videos representing around 3958 different video sequences. Moreover, those videos have been generated optimizing different parameters such as the luminosity or the contrast.

The point of the paper is not to present yet another variability language. We rather want to highlight the specific requirements we faced throughout the project, in the video domain, leading to the design and use of existing (or novel) variability constructs. Our experience, as others [49, 48, 6], question the existence of a one-size-fits-all variability language applicable in any industry. On the one hand, some common needs for modelling variability are more and more becoming apparent (e.g., support for attributes and multi-features [8, 23, 57]). On the other hand, some language constructs (visibility conditions, deltas, or meta-data to control the reasoning, etc.) are quite specific to subject systems and particular domains. Overall, it calls to further investigate the design space of variability languages as well as mechanisms that would support the customization of variability languages. From that perspective, we concur with Sepulveda *et al.* [13] that additional empirical studies are needed.

In our research, we have not evaluated dimensions of the VM language like learnability, readability or productivity. A research direction for future work is to conduct user experiments with VM.

Variability is gaining momentum in an increasing amount of domains and applications. The synthesis of video variants is a positive experience and an additional illustration. It perhaps explains the diversity of existing techniques, practices, tools, and languages for capturing variability.

We are now in the process of launching a very large-scale testing campaign over thousands of video variants – something clearly impossible at the beginning of the project (i.e., without variability support). As future work, we plan to investigate the effectiveness of sampling techniques [69–74] (e.g., pair-wise criterion) in our specific context and domain. We also plan to apply statistical methods for assessing vision algorithms and eventually building prediction models of their non functional properties [75–77].

Acknowledgements

This work was financed by the project MOTIV of the Direction Générale de l'Armement (DGA) - Ministère de la Défense, France. We thank all participants of the project. Special thanks to Pierre Romenteau from

InPixal (Rennes, France) for his continuous feedbacks and his joint development for synthesizing video variants.

References

1. M. Svahnberg, J. van Gurp, J. Bosch, A taxonomy of variability realization techniques: Research articles, Softw. Pract. Exper. 35 (8) (2005) 705–754. doi:http://dx.doi.org/10.1002/spe.v35:8.
2. S. Apel, D. Batory, C. Kästner, G. Saake, Feature-Oriented Software Product Lines: Concepts and Implementation, Springer-Verlag, 2013.
3. Product Line Hall of Fame, http://www.splc.net/fame.html.
4. T. Fogdal, H. Scherrebeck, J. Kuusela, M. Becker, B. Zhang, Ten years of product line engineering at danfoss: lessons learned and way ahead, in: Proceedings of the 20th International Systems and Software Product Line Conference, SPLC 2016, Beijing, China, September 16-23, 2016, 2016, pp. 252–261.
5. K. Pohl, G. Böckle, F. van der Linden, Software Product Line Engineering - Foundations, Principles, and Techniques, Springer, 2005.
6. Berger, Thorsten and Rublack, Ralf and Nair, Divya and Atlee, Joanne M. and Becker, Martin and Czarnecki, Krzysztof and Wasowski, Andrzej, A survey of variability modeling in industrial practice, in: VaMoS'13, ACM, 2013.
7. K. Czarnecki, S. Helsen, U. W. Eisenecker, Formalizing cardinality-based feature models and their specialization, Software Process: Improvement and Practice 10 (1) (2005) 7–29.
8. M. Cordy, P.-Y. Schobbens, P. Heymans, A. Legay, Beyond boolean product-line model checking: dealing with feature attributes and multi-features, in: ICSE'13, 2013, pp. 472–481.
9. D. Benavides, S. Segura, A. R. Cortés, Automated analysis of feature models 20 years later: A literature review, Inf. Syst. 35 (6) (2010) 615–636.
10. J. A. Galindo, M. Alférez, M. Acher, B. Baudry, D. Benavides, A variability-based testing approach for synthesizing video sequences, in: International Symposium on Software Testing and Analysis, ISSTA'14, San Jose, CA, USA - July 21 - 26, 2014, 2014, pp. 293–303. doi:10.1145/2610384.2610411.
    URL http://doi.acm.org/10.1145/2610384.2610411
11. M. Acher, M. Alférez, J. A. Galindo, P. Romenteau, B. Baudry, Vivid: a variability-based tool for synthesizing video sequences, in: 18th International Software Product Lines Conference - Companion Volume for Workshop, Tools and Demo papers, SPLC '14, Florence, Italy, September 15-19, 2014, 2014, pp. 143–147. doi:10.1145/2647908.2655981.
    URL http://doi.acm.org/10.1145/2647908.2655981
12. A. Hubaux, A. Classen, M. Mendonça, P. Heymans, A preliminary review on the application of feature diagrams in practice, in: VaMoS, 2010, pp. 53–59.
13. S. Sepulveda, A. Cravero, C. Cachero, Requirements modelling languages for software product lines: a systematic literature review, Information and Software Technology (2015) –doi:http://dx.doi.org/10.1016/j.infsof.2015.08.007.
    URL http://www.sciencedirect.com/science/article/pii/S0950584915001494
14. S. Nadi, S. Krüger, Variability modeling of cryptographic components: Clafer experience report, in: Proceedings of the Tenth International Workshop on Variability Modelling of Software-intensive Systems, Salvador, Brazil, January 27 - 29, 2016, 2016, pp. 105–112.
15. T. Berger, D. Nair, R. Rublack, J. M. Atlee, K. Czarnecki, A. Wasowski, Three cases of feature-based variability modeling in industry, in: Model-Driven Engineering Languages and Systems - 17th International Conference, MODELS 2014, Valencia, Spain, September 28 - October 3, 2014. Proceedings, 2014, pp. 302–319.
16. P. Temple, J. A. Galindo Duarte, M. Acher, J.-M. Jézéquel, Using machine learning to infer constraints for product lines, in: Software Product Line Conference (SPLC'16), Beijing, China, 2016.
17. J. R. Parker, Algorithms for image processing and computer vision, Wiley. com, 2010.
18. J. Ponce, D. Forsyth, E.-p. Willow, S. Antipolis-Méditerranée, R. d'activité RAweb, L. Inria, I. Alumni, Computer vision: a modern approach, Computer 16 (2011) 11.
19. H. Zhang, J. E. Fritts, S. A. Goldman, Image segmentation evaluation: A survey of unsupervised methods, Computer Vision and Image Understanding 110 (2) (2008) 260–280.
20. S. Oh, A. Hoogs, A. Perera, N. Cuntoor, C.-C. Chen, J. T. Lee, S. Mukherjee, J. K. Aggarwal, H. Lee, L. Davis, E. Swears, X. Wang, Q. Ji, K. Reddy, M. Shah, C. Vondrick, H. Pirsiavash, D. Ramanan, J. Yuen, A. Torralba, B. Song, A. Fong, A. Roy-Chowdhury, M. Desai, A large-scale benchmark dataset for event recognition in surveillance video, in: Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition, CVPR '11, IEEE Computer Society, Washington, DC, USA, 2011, pp. 3153–3160. doi:10.1109/CVPR.2011.5995586.
    URL http://dx.doi.org/10.1109/CVPR.2011.5995586
21. D. Benavides, P. Trinidad, A. R. Cortés, S. Segura, Fama, in: Systems and Software Variability Management, 2013, pp. 163–171.
22. M. Acher, P. Collet, P. Lahire, R. B. France, Familiar: A domain-specific language for large scale management of feature models, Science of Computer Programming (SCP) 78 (6) (2013) 657–681.
23. K. Bak, K. Czarnecki, A. Wasowski, Feature and meta-models in clafer: Mixed, specialized, and coupled, in: SLE, 2010, pp. 102–122.
24. M. B. Cohen, M. B. Dwyer, J. Shi, Coverage and adequacy in software product line testing, Proceedings of the ISSTA 2006 workshop on Role of software architecture for testing and analysis - ROSATEA '06 (2006) 53–63doi:10.1145/1147249.1147257.
    URL http://portal.acm.org/citation.cfm?doid=1147249.1147257
25. S. Ida, S. Ketil, Technology research explained, Tech. rep. (2007).
26. M. Alférez, J. A. Galindo, M. Acher, B. Baudry, Modeling Variability in the Video Domain: Language and Experience Report, Rapport de recherche RR-8576, INRIA (Jul. 2014).
    URL http://hal.inria.fr/hal-01023159
27. J. Savolainen, M. Raatikainen, T. Männistö, Eight practical considerations in applying feature modeling for product lines, in: ICSR, 2011, pp. 192–206.

28. C. Nie, H. Leung, A survey of combinatorial testing, ACM Comput. Surv. 43 (2) (2011) 11.
29. S. Segura, J. A. Galindo, D. Benavides, J. A. Parejo, A. R. Cortés, Betty: benchmarking and testing on the automated analysis of feature models, in: VaMoS, 2012, pp. 63–71.
30. P.-Y. Schobbens, P. Heymans, J.-C. Trigaux, Feature diagrams: A survey and a formal semantics, in: RE, 2006, pp. 136–145.
31. M. Galster, D. Weyns, D. Tofan, B. Michalik, P. Avgeriou, Variability in software systems—a systematic literature review, IEEE Trans. Softw. Eng. 40 (3) (2014) 282–306. doi:10.1109/TSE.2013.56.
URL http://dx.doi.org/10.1109/TSE.2013.56
32. H. Eichelberger, K. Schmid, A systematic analysis of textual variability modeling languages, in: SPLC, 2013, pp. 12–21.
33. H. Eichelberger, K. Schmid, Mapping the design-space of textual variability modeling languages: a refined analysis, STTT 17 (5) (2015) 559–584. doi:10.1007/s10009-014-0362-x.
URL https://doi.org/10.1007/s10009-014-0362-x
34. U. M. of pure::variants, https://www.pure-systems.com/fileadmin/downloads/pure-variants/doc/pv-user-manual.pdf.
35. D. Beuche, Using pure: variants across the product line lifecycle, in: Proceedings of the 20th International Systems and Software Product Line Conference, SPLC 2016, Beijing, China, September 16-23, 2016, 2016, pp. 333–336. doi:10.1145/2934466.2962729.
URL http://doi.acm.org/10.1145/2934466.2962729
36. http://www.biglever.com/solution/product.html.
37. C. W. Krueger, P. C. Clements, Systems and software product line engineering with gears from biglever software, in: 18th International Software Product Lines Conference - Companion Volume for Workshop, Tools and Demo papers, SPLC '14, Florence, Italy, September 15-19, 2014, 2014, pp. 121–125. doi:10.1145/2647908.2655976.
URL http://doi.acm.org/10.1145/2647908.2655976
38. K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, A. S. Peterson, Feature-oriented domain analysis (foda) feasibility study, Tech. rep., DTIC Document (1990).
39. A. van Deursen, P. Klint, Domain-specific language design requires feature descriptions, Journal of Computing and Information Technology 10 (1) (2002) 1–17.
40. M. Mendonça, M. Branco, D. D. Cowan, S.p.l.o.t.: software product lines online tools, in: OOPSLA Companion, 2009, pp. 761–762.
41. M. Rosenmüller, N. Siegmund, T. Thüm, G. Saake, Multi-dimensional variability modeling, in: VaMoS, 2011, pp. 11–20.
42. V. Weber, Utfm - a next generation language and tool for feature modeling, Ph.D. thesis, Faculty of Electrical Engineering, Mathematics and Computer Science of the University of Twente (August 2014).
URL http://essay.utwente.nl/65854/
43. C. Quinton, D. Romero, L. Duchien, Cardinality-based feature models with constraints: a pragmatic approach, in: 17th International Software Product Line Conference, SPLC 2013, Tokyo, Japan - August 26 - 30, 2013, 2013, pp. 162–166. doi:10.1145/2491627.2491638.
URL http://doi.acm.org/10.1145/2491627.2491638
44. A. Abele, Y. Papadopoulos, D. Servat, M. Törngren, M. Weber, The CVM framework - A prototype tool for compositional variability management, in: Fourth International Workshop on Variability Modelling of Software-Intensive Systems, 2010, pp. 101–105.
URL http://www.vamos-workshop.net/proceedings/VaMoS_2010_Proceedings.pdf
45. A. Classen, Q. Boucher, P. Heymans, A text-based approach to feature modelling: Syntax and semantics of tvl, Sci. Comput. Program. 76 (12) (2011) 1130–1143.
46. A. Murashkin, M. Antkiewicz, D. Rayside, K. Czarnecki, Visualization and exploration of optimal variants in product line engineering, in: 17th International Software Product Line Conference, SPLC 2013, Tokyo, Japan - August 26 - 30, 2013, 2013, pp. 111–115. doi:10.1145/2491627.2491647.
URL http://doi.acm.org/10.1145/2491627.2491647
47. M.-O. Reiser, Core concepts of the Compositional Variability Management framework (CVM) –A Practitioner's Guide, Technical report Bericht-Nr. 2009-16, Technische Universitat Berlin (2009).
URL http://hal.inria.fr/hal-01023159
48. T. Berger, S. She, R. Lotufo, A. Wasowski, K. Czarnecki, A study of variability models and languages in the systems software domain, IEEE Trans. Software Eng. 39 (12) (2013) 1611–1640.
49. C. Dumitrescu, R. Mazo, C. Salinesi, A. Dauron, Bridging the gap between product lines and systems engineering: an experience in variability management for automotive model based systems engineering, in: T. Kishi, S. Jarzabek, S. Gnesi (Eds.), SPLC, ACM, 2013, pp. 254–263.
50. C. Dumitrescu, P. Tessier, C. Salinesi, S. Gérard, A. Dauron, R. Mazo, Capturing variability in model based systems engineering, in: M. Aiguier, F. Boulanger, D. Krob, C. Marchal (Eds.), CSDM, Springer, 2013, pp. 125–139.
51. G. Mussbacher, J. Araújo, A. Moreira, D. Amyot, Aourn-based modeling and analysis of software product lines, Software Quality Journal 20 (3-4) (2012) 645–687.
52. P. Hofman, T. Stenzel, T. Pohley, M. Kircher, A. Bermann, Domain specific feature modeling for software product lines, in: SPLC (1), 2012, pp. 229–238.
53. J. Gray, K. Fisher, C. Consel, G. Karsai, M. Mernik, J.-P. Tolvanen, Dsls: the good, the bad, and the ugly, in: G. E. Harris (Ed.), OOPSLA Companion, ACM, 2008, pp. 791–794.
54. M. Völter, E. Visser, Product line engineering using domain-specific languages, in: Software Product Lines - 15th International Conference, SPLC 2011, 2011, pp. 70–79.
55. M. Voelter, S. Benz, C. Dietrich, B. Engelmann, M. Helander, L. C. L. Kats, E. Visser, G. Wachsmuth, DSL Engineering - Designing, Implementing and Using Domain-Specific Languages, dslbook.org, 2013.
56. A. S. Karatas, H. Oguztüzün, A. H. Dogru, From extended feature models to constraint logic programming, Sci. Comput. Program. 78 (12) (2013) 2295–2312.
57. W. Zhang, H. Yan, H. Zhao, Z. Jin, A bdd-based approach to verifying clone-enabled feature models' constraints and customization, in: ICSR, 2008, pp. 186–199.

58. J. White, B. Dougherty, D. C. Schmidt, D. Benavides, Automated reasoning for multi-step feature model configuration problems, in: Proceedings of the 13th International Software Product Line Conference, SPLC '09, Carnegie Mellon University, Pittsburgh, PA, USA, 2009, pp. 11–20.
URL http://dl.acm.org/citation.cfm?id=1753235.1753238
59. J. White, D. Benavides, D. C. Schmidt, P. Trinidad, B. Dougherty, A. R. Cortés, Automated diagnosis of feature model configurations, Journal of Systems and Software 83 (7) (2010) 1094–1107. doi:10.1016/j.jss.2010.02.017.
URL https://doi.org/10.1016/j.jss.2010.02.017
60. R. DeMilli, A. J. Offutt, Constraint-based automatic test data generation, IEEE Transactions on Software Engineering 17 (9) (1991) 900–910.
61. S. Anand, E. K. Burke, T. Y. Chen, J. Clark, M. B. Cohen, W. Grieskamp, M. Harman, M. J. Harrold, P. Mcminn, et al., An orchestrated survey of methodologies for automated software test case generation, Journal of Systems and Software 86 (8) (2013) 1978–2001.
62. A. Hervieu, D. Marijan, A. Gotlieb, B. Baudry, Practical minimization of pairwise-covering test configurations using constraint programming, Information and Software Technology 71 (2016) 129–146.
63. G. Perrouin, S. Sen, J. Klein, B. Baudry, Y. Le Traon, in: Proc. of the International Conference on Software Testing (ICST), Paris, France, pp. 459–468.
64. S. Wang, S. Ali, A. Gotlieb, M. Liaaen, Automated product line test case selection: industrial case study and controlled experiment, Software and System Modeling 16 (2) (2017) 417–441. doi:10.1007/s10270-015-0462-4.
URL https://doi.org/10.1007/s10270-015-0462-4
65. S. Wang, D. Buchmann, S. Ali, A. Gotlieb, D. Pradhan, M. Liaaen, Multi-objective test prioritization in software product line testing: An industrial case study, in: Proceedings of the 18th International Software Product Line Conference - Volume 1, SPLC '14, ACM, New York, NY, USA, 2014, pp. 32–41. doi:10.1145/2648511.2648515.
URL http://doi.acm.org/10.1145/2648511.2648515
66. M. Grochtmann, K. Grimm, Classification trees for partition testing, Softw. Test., Verif. Reliab. 3 (2) (1993) 63–82. doi:10.1002/stvr.4370030203.
URL https://doi.org/10.1002/stvr.4370030203
67. S. Moisan, J.-P. Rigault, M. Acher, P. Collet, P. Lahire, Run time adaptation of video-surveillance systems: A software modeling approach, in: International Conference on Computer Vision Systems (ICVS'11), 2011, pp. 203–212.
68. M. Acher, P. Collet, P. Lahire, S. Moisan, J.-P. Rigault, Modeling variability from requirements to runtime, in: ICECCS, 2011, pp. 77–86.
69. C. Yilmaz, M. B. Cohen, A. A. Porter, Covering arrays for efficient fault characterization in complex configuration spaces, Software Engineering, IEEE Transactions on 32 (1) (2006) 20–34.
70. M. B. Cohen, M. B. Dwyer, J. Shi, Coverage and adequacy in software product line testing, in: Proceedings of the ISSTA 2006 workshop on Role of software architecture for testing and analysis, ACM, 2006, pp. 53–63.
71. M. F. Johansen, Ø. Haugen, F. Fleurey, An algorithm for generating t-wise covering arrays from large feature models, in: 16th International Software Product Line Conference, SPLC '12, 2012, pp. 46–55.
72. S. Apel, A. von Rhein, P. Wendler, A. Größlinger, D. Beyer, Strategies for product-line verification: Case studies and experiments, in: ICSE'13, IEEE, 2013.
73. C. Henard, M. Papadakis, G. Perrouin, J. Klein, P. Heymans, Y. L. Traon, Bypassing the combinatorial explosion: Using similarity to generate and prioritize t-wise test configurations for software product lines, IEEE Trans. Software Eng. 40 (7) (2014) 650–670.
74. T. Thüm, S. Apel, C. Kästner, I. Schaefer, G. Saake, A classification and survey of analysis strategies for software product lines, ACM Computing Surveys (CSUR) 47 (1) (2014) 6.
75. N. Siegmund, M. Rosenmüller, M. Kuhlemann, C. Kästner, S. Apel, G. Saake, Spl conqueror: Toward optimization of non-functional properties in software product lines, Software Quality Journal 20 (3) (2011) 487–517. doi:http://dx.doi.org/10.1007/s11219-011-9152-9.
URL http://www.springerlink.com/content/ax788q46h1702j34/
76. J. Guo, K. Czarnecki, S. Apel, N. Siegmund, A. Wasowski, Variability-aware performance prediction: A statistical learning approach, in: 28th IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE, IEEE, Silicon Valley, California, USA, 2013.
77. S. Sobernig, S. Apel, S. Kolesnikov, N. Siegmund, Quantifying structural attributes of system decompositions in 28 feature-oriented software product lines: An exploratory study., Empirical Software Engineering 21 (4).