Defining Categorical Reasoning of Numerical Feature Models with Feature-Wise and Variant-Wise Quality Attributes

Daniel-Jesus Munoz
ITIS Software, Universidad de Málaga, Andalucía Tech
Málaga, Spain
danimg@lcc.uma.es

Mónica Pinto
ITIS Software, Universidad de Málaga, Andalucía Tech
Málaga, Spain
pinto@lcc.uma.es

Dilian Gurov
KTH Royal Institute of Technology
Stockholm, Sweden
dilian@kth.se

Lidia Fuentes
ITIS Software, Universidad de Málaga, Andalucía Tech
Málaga, Spain
lff@lcc.uma.es

ABSTRACT

Automatic analysis of variability is an important stage of *Software Product Line* (SPL) engineering. Incorporating quality information into this stage poses a significant challenge. However, quality-aware automated analysis tools are rare, mainly because in existing solutions variability and quality information are not unified under the same model.

In this paper, we make use of the *Quality Variability Model* (QVM), based on *Category Theory* (CT), to redefine reasoning operations. We start defining and composing the six most common operations in SPL, but now as quality-based queries, which tend to be unavailable in other approaches. Consequently, QVM supports interactions between variant-wise and feature-wise quality attributes. As a proof of concept, we present, implement and execute the operations as lambda reasoning for CQL IDE – the state-of-the-art CT tool.

CCS CONCEPTS

• **Software and its engineering → Abstraction, modeling and modularity**; **Software product lines**; **Software performance**; *Requirements analysis*; • **Theory of computation → Automated reasoning**; • **Computing methodologies → Representation of mathematical objects**.

KEYWORDS

category theory, quality attribute, extended feature model, automated reasoning, numerical features

ACM Reference Format:
Daniel-Jesus Munoz, Mónica Pinto, Dilian Gurov, and Lidia Fuentes. 2022. Defining Categorical Reasoning of Numerical Feature Models with Feature-Wise and Variant-Wise Quality Attributes. In *26th ACM International Systems and Software Product Line Conference - Volume B (SPLC '22), September 12–16, 2022, Graz, Austria.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3503229.3547057

SPLC '22, September 12–16, 2022, Graz, Austria

ACM ISBN 978-1-4503-9206-8/22/09…$15.00
https://doi.org/10.1145/3503229.3547057

1 INTRODUCTION

The features of *Software Product Lines* (SPLs) and their relationships are represented as *Variability Models* (VMs) [25], being Feature Models (FMs) the de facto standard. How to include quality information in these VMs, and how to perform quality analyses of SPLs, are two of the main challenges that remain open. Unfortunately, the lack of a broad consensus about how to embed *Quality Attributes* (QAs) in a VM has led to entirely different solutions, which in most cases are not materialised in a tool [18, 32].

Consequently, quality-aware reasoning (e.g., cost or energy consumption) is not directly supported by current SPL solvers and approaches, and hence, the literature eludes this issue [2]. Many workarounds assume that all QAs can be represented as attributes linked to single features (i.e., *feature-wise* approach) and that the product quality can be computed with simple equations such as addition [10]. Or, rather, they use a combination of solvers connected with external applications such as databases storing QA values or use domain learning procedures to predict the quality of configurations [15], all separate from the VM. All these works propose non-unified solutions, are not efficient, and with limited reasoning capabilities [21]. Most importantly, QAs whose measures (e.g., the energy consumption of a piece of code) depend on many features (e.g., CPU, OS, programming language), cannot be quantified at the VM feature level and need to be measured at the configuration level, i.e., they are *product variant-wise QAs* [26]. Indeed, in [26] authors claim that using feature attributes to store QAs is only valid for feature-wise QAs (e.g., the Cost associated with a feature). So, VMs and analysis operations should include quality information supporting both feature-wise and product variant-wise approaches. Unfortunately, state-of-the-art tools do not adequately support this yet.

The SPL community need to achieve a formal representation that unifies VMs and quality models (QMs), extensible and fully supported by an automated reasoner. With this objective, we leveraged the modelling and reasoning flexibility of *Category Theory* (CT) which captures the common aspects of seemingly dissimilar algebraic structures [1]. We published a preliminary work at [19]: an open-source SPL framework that unifies numerical VMs (i.e., supports both boolean and numerical features) with QAs related as

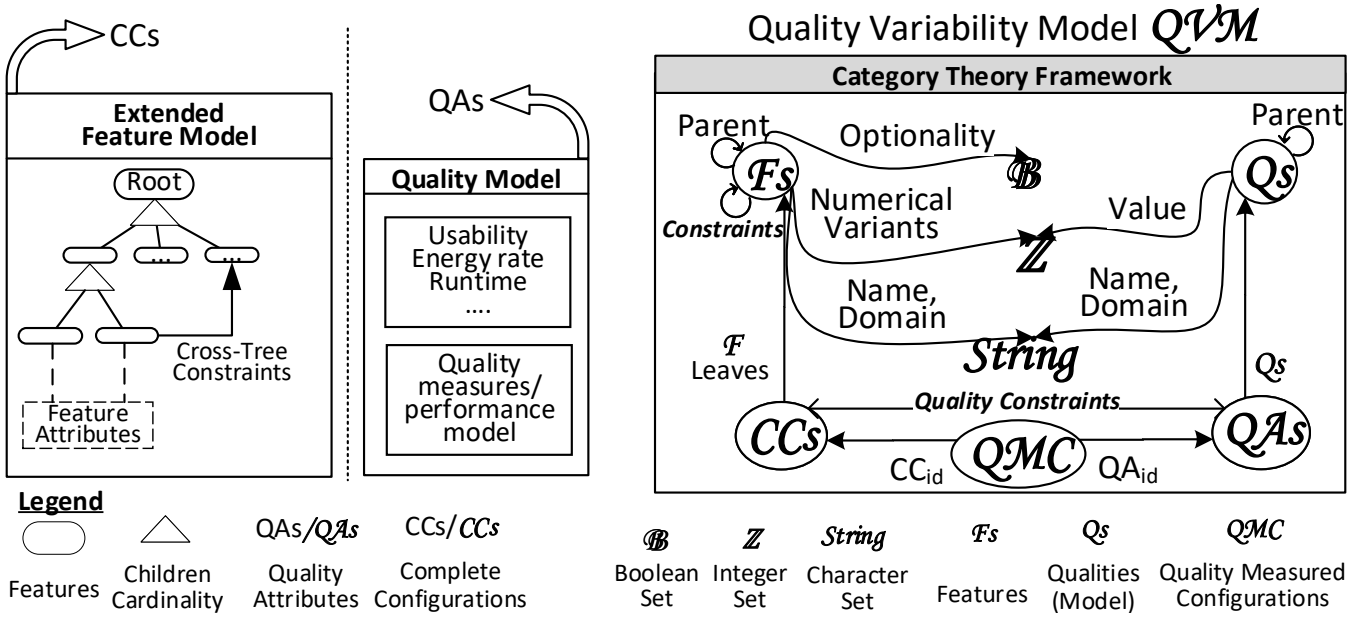

Figure 1: Unification of Extended Variability Models and Quality Models into a Category

an SPL category. Both VM's features and QAs are CT objects, while hierarchical cross-tree constraints and QAs are CT arrows. But, that work needed to be adjusted for the complete set of operations that exploit any type of quality information instantiated in the SPL category.

Consequently, in this work, we are defining the new *Quality Variability Model* (QVM), which directly supports quality-reasoning operations on SPL configurations. Extending the published proposal with Section 2, we now allow developers to perform richer variability analyses focused on quality, using common operations on VMs.

Having reviewed the state-of-the-art tools and literature in Section 3, we identified six basic operations that can be combined and extended into more complex ones. Two of them (*satisfiability* and *counting*) are considered essential self-analysis operations over VMs. The remaining ones (*filtered search*, *limited search*, *randomise* and an *operation combining feature-wise and variant-wise interactions*) are standard operations for generating configurations, which we reformulated to include QAs-related queries to be resolved natively by a CT framework (e.g., count the number of configurations within an energy consumption rate interval without listing them, or generate ten configurations with an energy consumption lower than a particular amount).

The main contribution of this work is that these operations can natively analyse the functional variability and its correlations with the non-functional (i.e., quality) requirements for a VM enriched with quality measures, i.e., our QVM. In the proof of concept of Section 6, we first present our lambda-function implementation for the CT editor *Categorical Query Language Integrated Development Environment* (CQL IDE) to show its potential for SPL stakeholders. CQL is a user-friendly IDE that allows categories to be defined as functional programming databases which are faster and more flexible than classical databases [6]. Then, and for illustrative purposes, we analyse a full scenario of a use case from the intelligent network and edge computing domain, specified through a variability model of network functionalities (i.e., Virtual Network Functions or VNFs) enriched with quality measurements of energy footprint and monetary cost. Additionally, we measured reasoning times of our implementation in CQL IDE. Its results indicate that CQL IDE flexibility and scalability for modelling and reasoning are promising as a tool for both the CT and the SPL communities.

2 QUALITY VARIABILITY MODEL *CATEGORY*

This section provides some background on the fundamentals of CT and present the unified **Q**uality ***V**ariability **M**odel* (QVM).

2.1 Fundamentals of Category Theory

Category Theory (CT) is an algebraic theory of mathematical structures [1]. It allows to capture and relate similar aspects of structures while abstracting from the individual specifics of their dissimilarities. A category C represents spaces as a collection of *objects* related to one another via *arrows* (i.e., *morphisms*). Two examples are the categories $\mathcal{V}ec$, where the objects are vector spaces and the arrows are linear maps, and $\mathcal{S}et$ where objects are sets and arrows are functions from one set to another. The main concepts of CT are:

- **Object**: a structured class $X \in \mathrm{Ob}(C)$, graphically depicted as a node $\bullet^X$.
- **Arrow**: a structure-preserving function $a \in \mathrm{Arr}(C)$ with source and target objects $X = src(a)$ and $Y = tgt(a)$, respectively, depicted $\overset{X}{\bullet} \overset{a}{\longrightarrow} \overset{Y}{\bullet}$.
  - **Identity**: for every $X \in \mathrm{Ob}(C)$, we have the arrow $\overset{X}{\bullet} \overset{id}{\longrightarrow} \overset{X}{\bullet}$.

  - **Composition**: if $\overset{X}{\bullet} \xrightarrow{a_1} \overset{Y}{\bullet}$ and $\overset{Y}{\bullet} \xrightarrow{a_2} \overset{Z}{\bullet}$, then $\overset{X}{\bullet} \xrightarrow{a_2 \circ a_1} \overset{Z}{\bullet}$. Composition is associative, i.e., $a_1 \circ (a_2 \circ a_3) = (a_1 \circ a_2) \circ a_3$.
- **Category**: consists of $\mathrm{Ob}(C) \cup \mathrm{Arr}(C)$ in a labelled directed graph.
- **Functor**: a process F between categories $C = src(F)$ and $D = tgt(F)$, depicted $\overset{C}{\bullet} \xrightarrow{F} \overset{D}{\bullet}$, which preserves identity and function composition.

Also, we shall introduce algebraic data integration CT concepts [6]:

- **Path**: a finite sequence of composed arrows: $\overset{X_0}{\bullet} \xrightarrow{a_1} \overset{X_1}{\bullet} \cdots \overset{X_{n-1}}{\bullet} \xrightarrow{a_n} \overset{X_n}{\bullet}$.
- **Element**: for $X \in \mathrm{Ob}(C)$, a *generalised element* of X is a morphism $\overset{U}{\bullet} \xrightarrow{elem} \overset{X}{\bullet}$, where U is a select "unit" object.
- **Instance**: a set-valued functor **Inst** that assigns values to elements.

2.2 Unifying Variability and Quality in a Categorical Model

In [19], we presented a preliminary SPL framework that unifies numerical VMs with QAs as a category where features and QAs are objects, and data-types, hierarchical relationships, and quality and feature constraints are arrows. We now extended that model to provide an alternative framework to abstract any QVM as a category $\mathcal{QVM}$. This framework allows QAs modelling at the feature-wise and also variant-wise levels. Hence, by supporting varian-wise QAs we solve the known challenge of complex QAs with many feature interactions – unfeasible to accurately distribute feature-wise by managing a complex aggregation function.

The framework and its equivalences are graphically represented in Figure 1, being the basis for the formalisation of the quality-aware operations further presented in this paper. Concretely, $\mathcal{QVM}$ comprises 3 data-type objects (i.e., $\mathbb{B}$ for Boolean, $\mathbb{Z}$ for integer, and $\mathcal{S}tring$ for characters sets) and 5 structured objects:

(1) $\mathcal{F}s$: It hosts the extended VM as arrows $\mathcal{F}s \rightarrow \mathcal{S}tring$ for the name and domain, $\mathcal{F}s \rightarrow \mathbb{B}$ for optionality, $\mathcal{F}s \rightarrow \mathbb{Z}$ for a numerical value, $\mathcal{F}s \rightarrow \mathcal{F}s$ for hierarchical (i.e., *Parent*) and cross-tree constraints, and additional arrows depending on the number of QAs at the feature-level.
(2) $\mathcal{Q}s$: It hosts the quality model as the arrows $\mathcal{Q}s \rightarrow \mathcal{S}tring$ for quality name and domain, $\mathcal{Q}s \rightarrow \mathbb{Z}$ for a quality value, and $\mathcal{Q}s \rightarrow \mathcal{Q}s$ for hierarchical and quality self-constraints.
(3) $\mathcal{CC}s$: It links the leaf features of specific complete configurations by $\mathcal{CC}s \rightarrow \mathcal{F}s$ instances. Please note that non-leaf features can be trace back to the root by *Parent* arrows.
(4) $\mathcal{QA}s$: It links QAs forming sets. It identifies sets of QAs as $\mathcal{QA}s \rightarrow \mathcal{Q}s$ instances and $\mathcal{QA}s \rightarrow \mathcal{QA}s$ constraints.
(5) $\mathcal{QMC}$: It is a list of **identifiers pairs** algebraically linking configurations with sets of variant-wise QAs. It comprises two arrows, $\mathcal{QMC} \rightarrow \mathcal{CC}s$ to identify each configuration and $\mathcal{QMC}s \rightarrow \mathcal{QA}s$ to identify their variant-wise QAs. Similarly, *Quality Constraints* are QAs limiting features of a CC, and vice-versa (e.g., WiFi requires Energy < 3 seconds).

3 RELATED WORK

In this section, we summarise the related work that has motivated this work. We list and compare academic and industrial tools, focusing on their support for variability modelling, quality modelling and automatic reasoning. Then, we discuss a motivating usage scenario to illustrate the shortcomings of existing works and the benefits of our approach.

Variability Modelling. Its first formalisation dates back to *Feature-oriented Domain Analysis* (FODA) [16], which proposes Feature Models (FMs) to specify a product family commonality and variability, and external solvers to automatically generate the product variants. FMs are represented as a rooted tree graph consisting of features such as Boolean variables and relationships (see Figure 1, left).

Relationships among features are specified in propositional logic, including tree (e.g., And, Or) and cross-tree constraints. Any SPL approach should provide at least the basic features and constraints defined in FODA. However, more than 45 extensions have been proposed for different needs [7]. The most relevant ones for our approach are: numerical features [20] that are features that can take integer or real numerical values, attributes [4] that allow associating a set of quality attributes to a feature, and complex constraints involving numerical features, attributes, etc. [14] concluded that current SPL tools do not support all the FM extensions and this prevents the adoption of them by SPL development processes.

As discussed in Section 2, the use of categories in our approach simplifies incorporating these extensions [19]. In this paper, we will show that QVM supports all the variability modelling extensions and a set of quality-aware operations.

Quality Modelling. There is no consensus on how to enrich variability models with quality information to reason about the quality of configurations [14, 30]. Typically, an SPL engineer would like to obtain the configurations with a quality attribute below a threshold (e.g., get configurations that consume less than 3 Joules), or to generate the best-qualified configuration (e.g. trade-off between energy consumption and performance). In summary, we consider two different approaches - one that extends traditional FMs, either with attributes or with an FM sub-tree, and one that defines a quality model and links quality measures with configurations.

In the first approach, quality information is provided at feature-wise level by attributes - i.e., each feature contributes individually to the system QA. In this case, quality measures are linked to single features using attributes [4] (e.g., the WiFi feature consumes 3 Joules). To compute the quality of feature configurations an aggregation function is defined, such as addition, minimum or maximum [26]. This approach is supported by Clafer, AAFM Python framework [8] (and the formerly FAMA), FeatureIDE, pure::variants and SPL Conqueror, all described in [14]. The most advanced extension formalisation is the *Universal Modelling Language* (UVL) [29]. However, it does not consider variant-wise QAs, nor there mentioned state-of-the-art solvers are UVL ready.

An alternative is to extend the FM by incorporating QA-specific features in a subtree. This is the case of QAMTool [33]. But [26] demonstrates that using attributes is only valid for feature-wise QAs, such as footprint or cost, where we can either measure a single

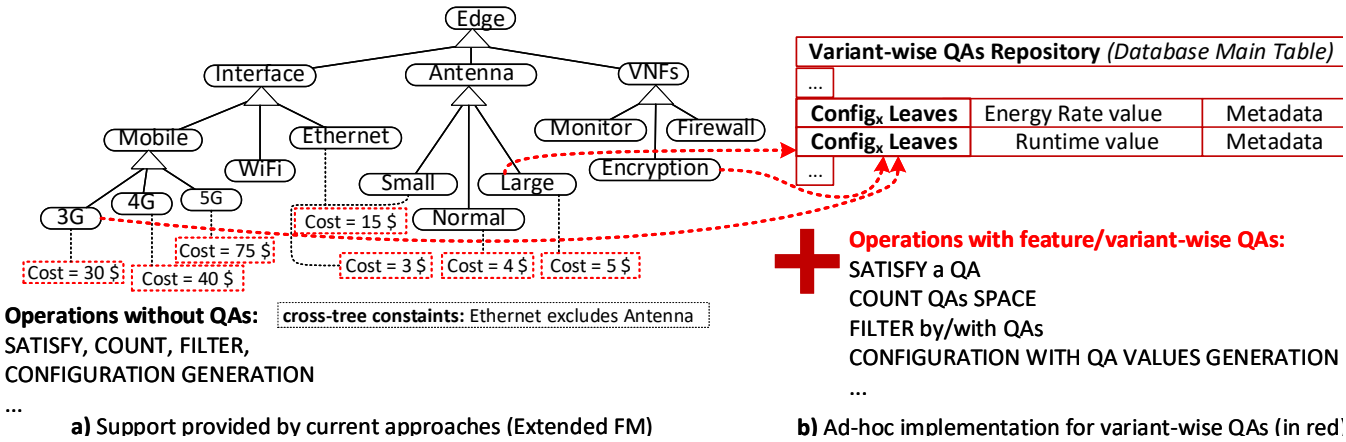

Figure 2: Motivating usage scenario

feature directly or infer the results of the measurement of a product variant to single features. Another problem is the difficulty of correctly distributing the quantitative value measured for a product variant to the attributes of the individual features [27], mainly due to not considering the feature interactions.

However, even defining better distribution algorithms, the problem remains for those QAs, such as performance and energy consumption, that do not make sense to measure at the feature-level (e.g., the energy consumption of WiFi feature depends on other features like data length, package size, distance to the access point, etc.). For these QAs, it is impossible to quantify their influence on individual features because the measurements have meaning only for a concrete configuration variant. These QAs are classified in [26] as variant-wise QAs, and several works consider that the set of measurements of a QA should be univocally linked to a configuration variant [9, 11, 13].

In some cases, an external quality model is defined (e.g., a goal model); and the QAs measurements are usually linked to the configurations through a database.

In [19], we highlighted the benefits and drawbacks of approaches to defining an external quality model, with two important conclusions: (1) most existing solutions are not directly compatible with automated quality-reasoning, and (2) SPL reasoning lacks a *"unified"* model that appropriately supports quality metrics. Sometimes these two approaches (i.e., feature-wise and variant-wise) are combined, as in QAMTool that uses the NFR framework [32] to externally represent QAs in a goal model and also an FM subtree to include quality information as part of the tree.

Our approach QVM is representative of the latest approach defining a unified model using a category theory framework with native support for quality reasoning. Also, QVM supports both the feature-wise and variant-wise approaches by associating at the configuration level the result of the feature-wise aggregate function. To the best of our knowledge, there is currently no proposal providing quality-aware reasoning of product variability, similar to QVM.

Automatic reasoning All tools offer some level of automatic reasoning. Some provide their implementations of all or a subset of the operations for self-analysis and basic generation of configurations defined in [3]. Others use a third-party modelling language that provides such support.

The number of provided operations considerably depends on the support provided by solvers. Regarding quality-aware operations, current approaches do not natively support the full set of quality-aware operations due to different reasons: incompatibility of SAT solvers with non-Boolean features such as quantified QAs [14], limitations of *Constraint Programming* (CP) solvers, which do not make assumptions on the algebraic properties of a solution space [12], limitations of the modelling language like the feature-level QAs in MOO Clafer tool [23], intrinsic complexity to adapt or extend the specific searching algorithms like counting in *Satisfiability Modulo Theories* (SMT) solvers [5], and/or scalability and other reasoning issues like in the extended VMs [22].

For instance, only Clafer, FeatureIDE and SPL Conqueror allow sampling configurations, and the process is not entirely random [20]. Moreover, these tools use quality information only to optimise configurations and to calculate the overall impact on a QA. *Thus, the main contribution of this paper is the uncovering of a quality-aware version of the operations typically defined for a VM, more than those typically supported in other approaches, as part of a unified QVM.*

4 MOTIVATING USAGE SCENARIOS

We now present a usage scenario in an $\mathcal{EDGE}$ case study.

As shown in Figure2.a), the system comprises 3 different configurable network interfaces (namely Mobile, WiFi and Ethernet), 3 sizes of Antennas, and 3 different Virtual Network Functions (VNFs) (namely Monitor, Firewall, Encryption). EDGE contains the cross-constraint that Ethernet excludes Antenna, since it is a wired system. Alice can select a VM approach discussed in Section 3, but only those supporting numerical features, attributes and complex constraints.

But Alice is also interested in performing certain quality analyses based on 3 QAs: Runtime, Energy Rate and Cost. Thus, she needs to relate the QA measurements with the features and/or configurations of the VM. Hence, there are two different scenarios:

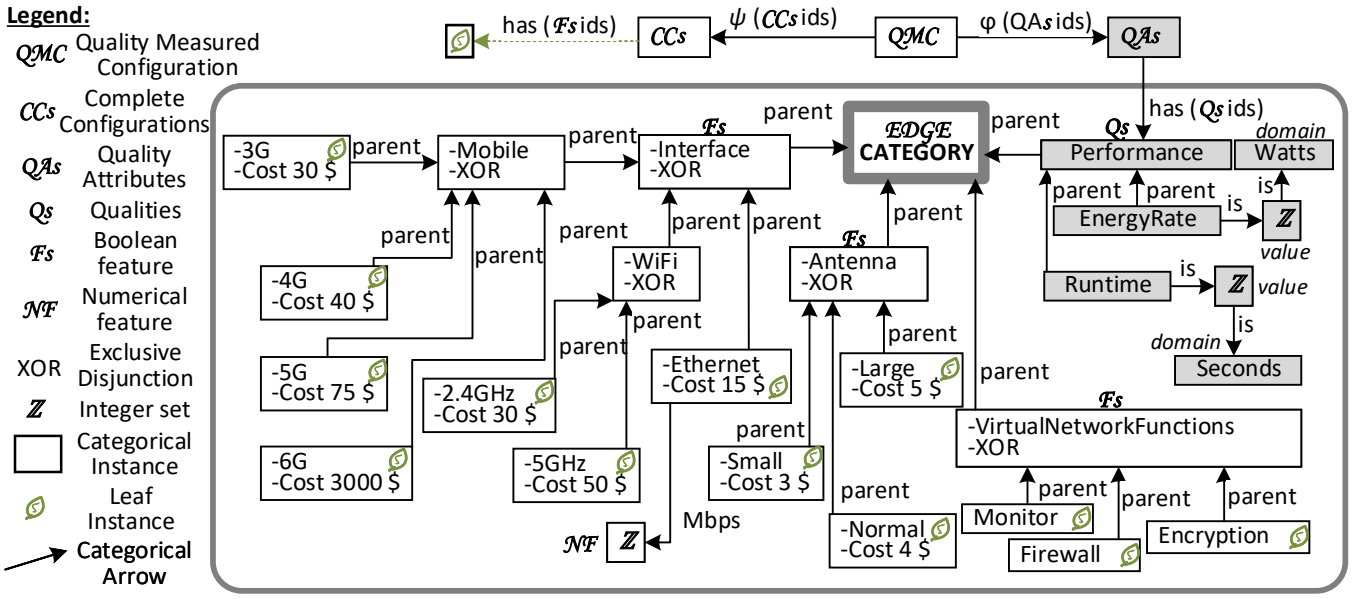

Figure 3: Olog representation of the partially instantiated $\mathcal{EDGE}$ SPL category

Scenario 1.1. Alice decides to use existing solutions:

- She can easily integrate the Cost measurements into the variability model as features' attributes because Cost is measured feature-wise (see Figure2.a)).
- However, she cannot do the same for the Runtime and Energy Rate, which are measured at the configuration level. This implies that she has to design, implement and use an ad-hoc external mechanism (e.g., a DB) to store these measures and link them to configurations in the FM (Figure2.b).

Scenario 1.2. Alice decides to use our approach:

- *We are offering Alice an integrated approach that will allow her to model variability with numerical features and complex constraints and, more importantly, to store both feature-wise and product variant-wise QAs using the same framework (Figure 3).*

However, although relating QA measurements and VMs is important, what Alice really needs is a list of quality-aware operations to perform the quality-analysis of her system. For instance, she first wants to check that the variability model is correct and that the reasoner can produce solutions that have Energy Rate and Runtime QAs information and thus she needs a SATISFY operation with regards to QAs. Then, she wants to count configurations, and not just the number of valid ones, but the number of configurations measured for both Runtime and Energy Rate values, needing a COUNT operation that considers QAs. Further, she wants to have a first idea of the kind of features and performance behaviour of $\mathcal{EDGE}$ instances and wants to bound the number of configurations to 5 configurations with Energy Rate, using for that a BOUND_RANDOM operation with QAs. This guides her to realise that the company only works with mobile networks interfaces, so the WiFi and Ethernet alternatives are meaningless.

As a consequence, she wants to filter the VM, having *Mobile* feature as a requirement (FILTER operation). She notices that the QA cost is indicated only at the feature level and that there are too many solutions yet. Consequently, she runs *FILTER* with constraints *Mobile* and *not 3G*, alongside *AGGREGATE* with the sum-aggregated function for Cost. At this point, she notices that some configurations are too costly for the performance that they provide. Finally, she runs again *FILTER_AGGREGATE* with an additional constraint: $(Cost \geq 40\,\$) \Rightarrow ((Energy\,Rate \leq 18\,Watts) \wedge (Runtime \leq 10\,Seconds))$. The solver returns several instances, showing her that a 5G network with a large antenna is optimal, besides for the *Monitor* virtual function, in which 4G networks with a small antenna behave equally with a fraction of the cost.

In order to perform the kind of analysis described above, there are two options:

Scenario 2.1. Alice decides to use existing solutions:

- Since operations implemented using current solvers (SATISFY, COUNT, FILTER, ...) do not take into account QAs, she has to provide her implementations of quality-aware operations (Figure 2.b)).
- In her implementations she needs to consider the different representations for feature-wise and variant-wise QAs.

Scenario 2.2. Alice decides to use our approach:

- *We are offering Alice a list of defined and implemented quality-aware operations using the native support provided by the framework used to model variability and QAs. We provide more of its details in the next section.*

5 QUALITY-AWARE REASONING OPERATIONS

In this section, we identify the basic operations to reason over QAs variant-wise. More complex operations can then be formed by combining and extending the basic ones. To illustrate these, we provide examples based on the instantiated $\mathcal{EDGE}$ category of Figure 3, which is graphically presented following the *olog* standard - the CT knowledge representation framework for real-world systems [28]. $\mathcal{EDGE}$ is our standard case study and comprises the categorical QVM of the edge-computing scenario described in Section 4.

5.1 Basic Operations on Quality Variability Models

We shall focus here on the operations typically needed to perform advanced reasoning over QVMs. We have selected 6 operations; all other operations are obtained as a composition of those. Operations *1.* and *2.* are analysis over the QVMs themselves, while the rest generate solution spaces of valued QAs.

(1) **Satisfiability, SATISFY** [31]: Check that there exists at least one measured configuration of a QVM with regards to the given QAs. For instance, is $\mathcal{EDGE}$ satisfiable? Checking satisfiability allows detecting inconsistent relational patterns.
(2) **Counting, COUNT** [20]: Count the instances of a QVM measured solution space without generating them. For example, how many configurations are in $\mathcal{EDGE}$, measured for *Runtime* in *Seconds*, and for *Energy Rate* in *Watts*?
(3) **Filtered search, FILTER** [14]: State advanced variability and QAs requirements over a QVM and generate its corresponding reduced solution space. For example, generate all $\mathcal{EDGE}$ configurations with their valued QAs matching the requirement:
(5G+ $\wedge$ (Runtime $\geq$ 2 Seconds)) $\Rightarrow$ (Energy Rate $\leq$ 1 Watt)
(4) **Bounded search, BOUND** [14]: Restrict the size of a QVM measured solution space. For example, generate 10 configurations of $\mathcal{EDGE}$ of the measured space for a certain QA.
(5) **Randomise, RANDOM** [20, 24]: Randomise the generation of a QVM measured solution space for any operation. For example, generate $\mathcal{EDGE}$ measured configurations starting from a random seed.
(6) **Aggregate, AGGREGATE**: Transform QAs at the feature level into QAs at the configuration level based on aggregation functions. E.g., the aggregated value of the cost of an $\mathcal{EDGE}$ configuration is the sum of the leaf features' costs.

6 IMPLEMENTATION IN CQL IDE AS LAMBDA FUNCTIONAL ALGORITHMS

In this section, we first provide details of our implementation of the six formalised $\mathcal{QVM}$ operations. They are presented as functional algorithms based on lambda functions.

For our implementation, we chose a state-of-the-art tool to model and reason over categories: the *Categorical Query Language* (CQL) IDE [1]. CQL is a functorial language used for functional programming with lambda functions; for background details, we kindly point the readers to [17].

[1]CQL IDE main website: https://www.categoricaldata.net/

Some details of CQL are:

- Basic data types and functions are defined as element type arrows between objects.
- A structured category is a *schema* of objects, and element or path types of arrows.
- A functor is a *query* over a source schema to a $\mathcal{RES}$ schema.
- A *literal instance* generates variables and assigns the values.
- The reasoning is an *eval instance* of a schema literal.

If any reasoning is undefined, the IDE halts and informs of the error; for instance, that $\mathcal{CCs}$ is not functorial due to a non-existing referenced feature. Otherwise, CQL IDE correctly computes what is specified in the instances and queries.

We now follow by presenting an overview of our implementation in CQL IDE of the identified QVM operations. We start with the Algorithm 1 implementing the SATISFY operation. Algorithm 1 is a single lambda function that returns *True* if any configuration (i.e., φ) and QA (i.e., ψ) relationship exists. As it is only computed over the $\mathcal{QMC}$ object, it completely abstracts from features or QAs names and values.

Algorithm 1: SATISFY for $\mathcal{QVMs}$ in CQL IDE

Input: Populated $\mathcal{QMC}$
Result: $\lambda(x \in \mathcal{QMC}) = (x.\varphi > 0) \wedge (x.\psi > 0)$;

Algorithm 2 implements operation COUNT. Similarly to Algorithm 1, it is composed of a single lambda function, but in this case, it adds 1 to a counter any time that there is a relationship between CCs and QAs in $\mathcal{QVMs}$.

Algorithm 2: COUNT for $\mathcal{QVMs}$ in CQL IDE

Input: Populated $\mathcal{QMC}$
Result: **Add**$(\lambda(x \in \mathcal{QMC}) : 1$ **If** $(x.\varphi > 0) \wedge (x.\psi > 0))$;

Algorithms 3 to 5 share some code, as they generate configurations with their respective QA values. Hence, in the three algorithms, besides $\mathcal{QMC}$, we also need $\mathcal{CCs}$ and $\mathcal{QAs}$ objects as inputs. On line 1 we initialise the result as the empty set *Res*, which is then filled in a *forall* loop iterating over each instance (i.e., relationship) of $\mathcal{QMC}$. Due to the *if* clause on line 3, just configurations related to QAs and vice-versa are considered in each iteration. Two lambda functions are filling *Res*. The first one is on line 4, and it retrieves the features associated with the configuration of a specific iteration. The second one is on line 5, and likewise retrieves the data of the set of qualities of a specific iteration. Finally, in each iteration, the features and QAs are copied to *Res* on lines 6 (7 for aggregation).

Continuing with the differences, for FILTER in Algorithm 3 we need to consider an additional input in the form of a set of features or quality constraints, which are then considered in the if clause of line 3 to filter the selected configurations and QAs. In Algorithm 4 we merged BOUND and RANDOM operations forming a random sampling. For that, we find two new inputs, a *Limit* (L) of solutions and a random *Seed* (S). On line 3, L restricts the number of tested loop iterations, and on Line 2 $\mathcal{QMC}$ is shuffled based on S.

Finally, in Algorithm 5 we have an extra input in the form of an aggregation *Function* (F) for QAs at the feature-level (e.g., addition

Algorithm 3: FILTER for $\mathcal{QVM}s$ in CQL IDE

Input: Populated $\mathcal{QMC}$, $\mathcal{CC}s$, $\mathcal{QA}s$, and ***C**onstraints* (C)

1 Res = λ [features, qas] : **forall** $x \in \mathcal{QMC}$ **do**
2 **if** $(x.\varphi > 0) \wedge (x.\psi > 0) \wedge (\mathcal{CC}s \notin C) \wedge (\mathcal{QA}s \notin C)$ **then**
3 features = $(\lambda y \in \mathcal{CC}s.feature : (\mathcal{CC}s.id = x.\varphi))$;
4 qas = $(\lambda z \in \mathcal{QA}s.qualities : (\mathcal{QA}s.id = x.\psi))$;
5 **end**
6 **end**

Result: Res

Algorithm 4: BOUND+RANDOM for $\mathcal{QVM}s$ in CQL IDE

Input: Populated $\mathcal{QMC}$, $\mathcal{CC}s$, $\mathcal{QA}s$, ***L**imit* (L), and ***S**eed* (S)

1 Res = λ [features, qas] : **forall** $x \in \mathit{SHUFFLE}(\mathcal{QMC}, S)$ **do**
2 **if** $(x.\varphi > 0) \wedge (x.\psi > 0) \wedge (\mathit{Res.size} \leq L)$ **then**
3 features = $(\lambda y \in \mathcal{CC}s.feature : (\mathcal{CC}s.id = x.\varphi))$;
4 qas = $(\lambda z \in \mathcal{QA}s.qualities : (\mathcal{QA}s.id = x.\psi))$;
5 **end**
6 **end**

Result: Res

Algorithm 5: AGGREGATE for $\mathcal{QVM}s$ in CQL IDE

Input: Populated $\mathcal{QMC}$, $\mathcal{CC}s$, $\mathcal{QA}s$, and ***F**unction* (F)

1 Res = λ [features, qas] : **forall** $x \in \mathcal{QMC}$ **do**
2 **if** $(x.\varphi > 0) \wedge (x.\psi > 0) \wedge (\mathit{Res.size} \leq L)$ **then**
3 features = $(\lambda y \in \mathcal{CC}s.feature : (\mathcal{CC}s.id = x.\varphi))$;
4 qas = $(\lambda\ F(attributes \in features),$
 $z \in \mathcal{QA}s.qualities : (\mathcal{QA}s.id = x.\psi))$;
5 **end**
6 **end**

Result: Res

of costs). Consequently, F is transformed into its correspondent lambda function on line 6, and adds additional calculated QAs to the results. ***This algorithm shows how feature-wise and variant-wise QAs can be treated uniformly in our approach. Notice that running the* AGGREGATE *operation before any of the other ones allows disposing of the feature-wise information at the configuration level.***

After testing them by running CQL IDE, its reasoning time averages to 0.13 seconds for Figure 3 $\mathcal{QVM}$. Consequently, we guess that CQL IDE is scalable due to its characteristic combination of reasoners: an automated theorem prover with Knuth-Bendix completion, and hashing, balanced binary search trees, and chasing algorithms. This selection comes from different fields: formal methods, SAT solving and relational algebra respectively.

7 PROOF-OF-CONCEPT

In this section, we revisit the usage scenario of the $\mathcal{EDGE}$ case study presented in Section 3 to show how Alice uses our approach to perform the QAs analysis of her $\mathcal{EDGE}$ scenario. First, Alice wants to check that the $\mathcal{QVM}$ is correct and that the reasoner can produce solutions. She runs SATISFY, and the CQL IDE returns an instance of $\mathcal{RES}$ with the value *True*, implying satisfiability without reasoning ambiguities.

Then, she runs COUNT to learn how many configurations the system has. CQL IDE returns an instance of $\mathcal{RES}$ with the value *63*. Further, she wants some idea of the kind of features and performance behaviour of $\mathcal{EDGE}$ instances, she runs a *BOUND_RANDOM* of 5 configurations. Since her company only works with mobile interfaces, she runs *FILTER* selecting the *Mobile* feature. CQL IDE returns 36 instances of $\mathcal{RES}$ comprising the same number of configurations with 2 valued QAs each, runtime and energy rate.

Since the QA cost is indicated only at the feature level, there are too many solutions. Consequently, she runs *FILTER* with constraints *Mobile* and *not 3G*, alongside *AGGREGATE* with the sum-aggregated function for cost. CQL IDE returns 27 instances of $\mathcal{RES}$ comprising the same number of configurations with the respective values of the 3 QAs – runtime, energy rate and cost.

As she notices that some configurations are too costly for the performance that they provide, she reruns *FILTER_AGGREGATE* with an additional constraint:
(Cost ≥ 40 $) ⇒ *((Energy Rate ≤ 18 Watts) ∧ (Runtime ≤ 10 Seconds))*.
CQL IDE returns 6 $\mathcal{RES}$ instances, showing her that a 5G network with a large antenna is the best option, besides for the *Monitor* virtual function, in which 4G networks with a small antenna behave equally with a fraction of the cost.

8 CONCLUSION

Before the unifying CT framework for SPLs [19], variability, quality, and the relationship between configurations and valued QAs have been modelled separately. This paper highlights the lack of methods and tools designed explicitly for QAs values at the feature and configuration levels. To address this, we identify the basic operations to quality-reason over QVMs: satisfiability, count, filter, bound, randomise, and aggregate.

Then, we present the categorical functional algorithms based on lambda functions for $\mathcal{QVM}s$. They are a guidance for CT reasoners implementation based on queries to a category. These operations over $\mathcal{QVM}s$ mainly operate on its object of binary relationships $\mathcal{QMC}$, as it provides complete configurations with valued QAs pairs. Additionally, we show how to combine operations to form complex ones. Further, we implement them in CQL IDE and perform a performance test for $\mathcal{EDGE}$ $\mathcal{QVM}$ with several casuistic.

CQL IDE $\mathcal{QMC}$ suggests scalability and hence a promising tool for the modelling and reasoning SPLs. We end with a usage scenario analysing the use of our approach in the $\mathcal{EDGE}$ case study.

In future work, we plan to properly perform a scalability test with large $\mathcal{QVM}s$ and more advanced queries (e.g., guided sampling, prediction). In addition, we are considering $\mathcal{QVM}$ implementations in other CT environments such as Haskell.

ACKNOWLEDGMENTS

Munoz, Pinto and Fuentes work is supported by the European Union's H2020 research and innovation programme under grant agreement DAEMON 101017109, by the projects co-financed by FEDER funds LEIA UMA18-FEDERJA-15, MEDEA RTI2018-099213-B-I00 and Rhea P18-FR-1081 and the PRE2019-087496 grant from the Ministerio de Ciencia e Innovación.

REFERENCES

[1] Michael Barr and Charles Wells. 1990. *Category theory for computing science*. Prentice Hall, Hoboken, New Jersey, USA.

[2] Rabih Bashroush, Muhammad Garba, Rick Rabiser, Iris Groher, and Goetz Botterweck. 2017. CASE Tool Support for Variability Management in SPLs. *ACM Comput. Surv.* 50, 1, Article 14 (March 2017), 45 pages. https://doi.org/10.1145/3034827

[3] David Benavides, Sergio Segura, and Antonio Ruiz-Cortés. 2010. Automated analysis of feature models 20 years later: A literature review. *Information Systems* 35, 6 (2010), 615 – 636. https://doi.org/10.1016/j.is.2010.01.001

[4] David Benavides, Pablo Trinidad, and Antonio Ruiz-Cortés. 2005. Automated Reasoning on Feature Models. In *Advanced Information Systems Engineering*, Oscar Pastor and João Falcão e Cunha (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 491–503.

[5] Mateus Borges, Quoc-Sang Phan, Antonio Filieri, and Corina S. Păsăreanu. 2017. Model-Counting Approaches for Nonlinear Numerical Constraints. In *NASA Formal Methods*, Clark Barrett, Misty Davies, and Temesghen Kahsai (Eds.). Springer International Publishing, Luxembourg, 131–138.

[6] Kristopher S Brown, David I Spivak, and Ryan Wisnesky. 2019. Categorical data integration for computational science. *Computational Materials Science* 164 (2019), 127–132.

[7] Lianping Chen, Muhammad Ali Babar, and Nour Ali. 2009. Variability Management in Software Product Lines: A Systematic Review. In *Proceedings of the 13th International Software Product Line Conference* (San Francisco, California, USA) *(SPLC '09)*. Carnegie Mellon University, USA, 81–90.

[8] José A. Galindo and David Benavides. 2020. A Python Framework for the Automated Analysis of Feature Models: A First Step to Integrate Community Efforts. In *Proceedings of the 24th ACM International Systems and Software Product Line Conference - Volume B (SPLC '20)*. ACM, New York, NY, USA, 52–55. https://doi.org/10.1145/3382026.3425773

[9] M. Glinz. 2007. On Non-Functional Requirements. In *15th IEEE International Requirements Engineering Conference (RE 2007)*. IEEE, Delhi, India, 21–26. https://doi.org/10.1109/RE.2007.45

[10] Jianmei Guo, Jia Hui Liang, Kai Shi, Dingyu Yang, Jingsong Zhang, Krzysztof Czarnecki, Vijay Ganesh, and Huiqun Yu. 2019. SMTIBEA: a hybrid multi-objective optimization algorithm for configuring large constrained software product lines. *Software & Systems Modeling* 18, 2 (2019), 1447–1466.

[11] Dilian Gurov, Bjarte M Østvold, and Ina Schaefer. 2011. A hierarchical variability model for software product lines. In *International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*. Springer, Springer, Luxembourg, 181–199.

[12] John N. Hooker. 2002. Logic, Optimization, and Constraint Programming. *INFORMS Journal on Computing* 14, 4 (2002), 295–321. arXiv:https://doi.org/10.1287/ijoc.14.4.295.2828

[13] Jose-Miguel Horcas, Mónica Pinto, and Lidia Fuentes. 2016. An automatic process for weaving functional quality attributes using a software product line approach. *Journal of Systems and Software* 112 (2016), 78 – 95.

[14] Jose-Miguel Horcas, Mónica Pinto, and Lidia Fuentes. 2019. Software Product Line Engineering: A Practical Experience. In *Proceedings of the 23rd International Systems and Software Product Line Conference - Volume A* (Paris, France) *(SPLC '19)*. ACM, New York, New York, USA, 164–176. https://doi.org/10.1145/3336294.3336304

[15] C. Kaltenecker, A. Grebhahn, N. Siegmund, and S. Apel. 2020. The Interplay of Sampling and Machine Learning for Software Performance Prediction. *IEEE Software* 37, 4 (2020), 58–66. https://doi.org/10.1109/MS.2020.2987024

[16] Kyo C Kang, Sholom G Cohen, James A Hess, William E Novak, and A Spencer Peterson. 1990. *Feature-oriented domain analysis (FODA) feasibility study*. Technical Report. Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst.

[17] Georg P Loczewski. 2018. *A++ and the Lambda Calculus: Principles of Functional Programming*. tredition, Berlin, Germany.

[18] Jens Meinicke, Thomas Thüm, Reimar Schröter, Fabian Benduhn, Thomas Leich, and Gunter Saake. 2017. *Quality Assurance for Feature Models and Configurations*. Springer International Publishing, Cham, 81–94. https://doi.org/10.1007/978-3-319-61443-4_8

[19] Daniel-Jesus Munoz, Dilian Gurov, Monica Pinto, and Lidia Fuentes. 2021. Category Theory Framework for Variability Models with Non-functional Requirements. In *Advanced Information Systems Engineering*, Marcello La Rosa, Shazia Sadiq, and Ernest Teniente (Eds.). Springer International Publishing, Cham, 397–413.

[20] Daniel-Jesus Munoz, Jeho Oh, Mónica Pinto, Lidia Fuentes, and Don Batory. 2019. Uniform Random Sampling Product Configurations of Feature Models That Have Numerical Features. In *Proceedings of the 23rd International Systems and Software Product Line Conference - Volume A* (Paris, France). ACM, New York, New York, USA, 289–301. https://doi.org/10.1145/3336294.3336297

[21] Daniel-Jesus Munoz, Mónica Pinto, and Lidia Fuentes. 2018. Finding correlations of features affecting energy consumption and performance of web servers using the HADAS eco-assistant. *Computing* 100, 11 (2018), 1155–1173.

[22] Lina Ochoa, Juliana Alves Pereira, Óscar González-Rojas, Harold Castro, and Gunter Saake. 2017. A Survey on Scalability and Performance Concerns in Extended Product Lines Configuration. In *Proceedings of the 11th Int. Workshop on VAMOS'2017* (Eindhoven, Netherlands). ACM, New York, New York, USA, 5–12.

[23] Rafael Olaechea, Steven Stewart, Krzysztof Czarnecki, and Derek Rayside. 2012. Modelling and Multi-Objective Optimization of Quality Attributes in Variability-Rich Software. In *Proceedings of the Fourth International Workshop on Nonfunctional System Properties in Domain Specific Modeling Languages* (Innsbruck, Austria). ACM, New York, New York, USA, Article 2, 6 pages. https://doi.org/10.1145/2420942.2420944

[24] Juliana Alves Pereira, Mathieu Acher, Hugo Martin, Jean-Marc Jézéquel, Goetz Botterweck, and Anthony Ventresque. 2021. Learning software configuration spaces: A systematic literature review. *Journal of Systems and Software* 182 (2021), 111044. https://doi.org/10.1016/j.jss.2021.111044

[25] Klaus Pohl, Günter Böckle, and Frank J van Der Linden. 2005. *Software product line engineering: foundations, principles and techniques*. Springer Science & Business Media, Luxembourg.

[26] Norbert Siegmund, Marko Rosenmüller, Martin Kuhlemann, Christian Kästner, Sven Apel, and Gunter Saake. 2012. SPL Conqueror: Toward Optimization of Non-Functional Properties in Software Product Lines. *Software Quality Journal* 20, 3–4 (sep 2012), 487–517. https://doi.org/10.1007/s11219-011-9152-9

[27] Norbert Siegmund, Stefan Sobernig, and Sven Apel. 2017. Attributed Variability Models: Outside the Comfort Zone. In *Proceedings of the 11th Joint Meeting on Foundations of Software Engineering* (Paderborn, Germany) *(ESEC/FSE 2017)*. ACM, New York, New York, USA, 268–278. https://doi.org/10.1145/3106237.3106251

[28] David I Spivak and Robert E Kent. 2012. Ologs: a categorical framework for knowledge representation. *PloS one* 7, 1 (2012), e24274.

[29] Chico Sundermann, Kevin Feichtinger, Dominik Engelhardt, Rick Rabiser, and Thomas Thüm. 2021. Yet another textual variability language? a community effort towards a unified language. In *Proceedings of the 25th ACM International Systems and Software Product Line Conference-Volume A*. ACM, New York, NY, USA, 136–147.

[30] Maurice H. ter Beek and Axel Legay. 2019. Quantitative Variability Modeling and Analysis. In *Proceedings of the 13th International Workshop on Variability Modelling of Software-Intensive Systems* (Leuven, Belgium) *(VAMOS '19)*. ACM, New York, New York, USA, Article 13, 2 pages. https://doi.org/10.1145/3302333.3302349

[31] Lanxin Yang, He Zhang, Haifeng Shen, Xin Huang, Xin Zhou, Guoping Rong, and Dong Shao. 2021. Quality Assessment in Systematic Literature Reviews: A Software Engineering Perspective. *Information and Software Technology* 130 (2021), 106397. https://doi.org/10.1016/j.infsof.2020.106397

[32] Anton Yrjönen and Janne Merilinna. 2009. Extending the NFR framework with measurable non-functional requirements. In *Proceedings of the 2nd International Workshop on Non-functional System Properties in Domain Specific Modeling Languages*, Marko Boškoviæ, Dragan Gaševiæ, Claus Pahl , and Bernhard Schätz (Eds.). ACM, New York, New York, USA, 0–14. 2nd International Workshop on Non-functional System Properties in Domain Specific Modeling Languages, NFPinDSML2009, NFPinDSML2009 ; Conference date: 04-10-2009 Through 04-10-2009.

[33] Guoheng Zhang, Huilin Ye, and Yuqing Lin. 2014. Quality attribute modeling and quality aware product configuration in software product lines. *Softw. Qual. J.* 22, 3 (2014), 365–401. https://doi.org/10.1007/s11219-013-9197-z