Category Theory Framework for Variability Models with Non-Functional Requirements

Daniel-Jesus Munoz[1,2], Dilian Gurov[3], Monica Pinto[1,2], and Lidia Fuentes[1,2]

[1] ITIS Software, Universidad de Málaga, Spain
[2] CAOSD, Departamento LCC, Universidad de Málaga, Andalucía Tech, Spain
{danimg, pinto, lff}@lcc.uma.es
[3] KTH Royal Institute of Technology, Stockholm, Sweden
dilian@kth.se

Abstract. In *Software Product Line* (SPL) engineering one uses *Variability Models* (VMs) as input to automated reasoners to generate optimal products according to certain *Quality Attributes* (QAs). Variability models, however, and more specifically those including numerical features (i.e., NVMs), do not natively support QAs, and consequently, neither do automated reasoners commonly used for variability resolution. However, those satisfiability and optimisation problems have been covered and refined in other relational models such as databases.
Category Theory (CT) is an abstract mathematical theory typically used to capture the common aspects of seemingly dissimilar algebraic structures. We propose a unified relational modelling framework subsuming the structured objects of VMs and QAs and their relationships into algebraic categories. This abstraction allows a combination of automated reasoners over different domains to analyse SPLs. The solutions' optimisation can now be natively performed by a combination of automated theorem proving, hashing, balanced-trees and chasing algorithms. We validate this approach by means of the edge computing SPL tool HADAS.

Keywords: numerical variability model · feature · non-functional requirement · quality attribute · category theory

1 Introduction

Variability Models [24] (VMs) are used for the design of highly configurable systems to represent their common and variable features, typically by means of a rooted tree graph with a set of constraints. These models employ two types of constraints: hierarchical (or tree) constraints, and cross-tree constraints, where the absence or value of some features instantiates or precludes other features (e.g., $feature_A$ implies/excludes $feature_B$). Variability models are the key asset in *Software Product Lines* (SPLs) [33], where valid configurations (i.e., solutions) are generated by reasoners called *solvers*, such as Choco [23] and Z3 [12], that take into account some external requirements. The most popular VMs are the *Feature Models* (FMs), but our problem formulation is agnostic of the VM type, so we will just refer generically to VMs throughout the rest of the paper.

One of the most valuable uses of VMs is the generation of optimal solutions [9] based on *Quality Attributes* (QAs) or *Non-Functional Requirements* (NFRs), e.g., to maximise performance or minimise energy consumption [29]. This becomes a difficult issue when tackling some emergent domains characterised by intensive variability such as *Internet of Things* (IoT) or *Edge Computing* (EC) systems [34], with variations at the hardware (e.g. sensors and edge devices), communication network (e.g. WiFi, BLE), application (e.g. filtering, mixing, collecting tasks) and infrastructure (e.g. virtualisation) dimensions. Regarding these application domains, one possible approach is to use VMs to specify the variability dimensions and use a solver to generate optimal application deployments in certain IoT/EC environments considering certain NFRs, such as latency or energy consumption. However, the standard VMs do not natively support non-functional properties, especially needed when one wants to express a relationship between one product and a NFR measured with a quality metric represented as a *measurement function* [18]. For example, the feature 'WiFi' of an IoT device consumes more or less energy, depending on the feature 'distance' to the Edge or Cloud device. The same concerns the automated reasoners for VMs that neither consider NFRs nor quality metrics as a built-in characteristic.

This problem has been tackled in different ways in recent years. For instance, *Extended VMs* [5] proposed to extend features with attached attributes, and they are used to indicate a QA value (e.g. energy_consumption, latency) of that specific feature. For example, one can express that the 'WiFi' feature consumes 'x' Joules or has a latency of 'y' Seconds, where Joules and Seconds are attributes. Extended VMs cannot represent that a certain QA is measured as a function of several features. But, QAs usually depend on several features representing a complete running product [32]. Another approach is to have independent VMs extended with a set of variables representing QA measurements and cross-model constraints as part of a constraint satisfaction problem [22], but not as part of the VM itself. This results in improper semantics, and variables and constraints overloading. A hybrid model that rudimentary links a VM with a QAs database is our previous work HADAS [31]. But again, the management of two different and interconnected models as well as two independent reasoners (i.e., Choco/database) is complex and computationally overloading.

Our goal is to extend the core definition of VMs with NFRs associated with product solutions, so that we can reason and generate optimal solutions that fit certain QAs. We propose *Category Theory* (CT) as a means to abstract and unify dissimilar relational models. We present a CT framework aiming to represent that: "each SPL product, defined as a set of 'n' features, is related to a set of QAs with concrete values that fulfil certain NFRs". In our CT framework, relational models are specified as objects and their relationships. As a result, we unify as a category: VMs, NFRs and QA metrics as measurement functions.

In the IoT/EC, it is common that some features (e.g. different message sizes) are *numerical features*; different numerical values can influence the energy consumption or the computation time NFRs. Therefore, our CT proposal considers to effortlessly represent and reason about *Numerical VMs* (NVMs), which

is not straightforward with traditional VMs [30]. This can be achieved only if they are part of the variability tree hierarchy when generating valid products. Contrarily to many of the existing Boolean VMs (e.g., FeatureIDE, Glencoe, and UVL), NVMs (e.g., Clafer [2] and Z3 [12]) additionally support numerical features and the relationships between them (i.e., variables and equations). However, the limitations of NVM solvers have prevented software developers from actively considering modelling numerical features [30]. Our contributions are:

1. A unified CT framework to model NVMs and QAs with NFRs and their relationships, and generate products as solutions with a sufficient quality.
2. As a proof of concept, we transform HADAS [31], a SPL to reason about energy consumption of IoT/Edge applications, into a category. We perform optimisation analyses with a combination of different reasoners, including a theorem prover and relational search algorithms, each one being able to reason at the same time about both VMs and quality metrics.

The paper is organised as follows. Section 2 describes VMs and QAs, while Section 3 defines CT and presents the framework to subsume VMs and QAs into categories. In Section 4 we test our approach by transforming an SPL tool into a category, and by reasoning about an EC case study. Section 5 reviews and discusses the pertinent related work, while Section 6 concludes with a summary highlighting the contributions and next steps of this research.

2 Motivation

Our goal is to use CT to define a joint model encompassing variability and NFRs modelling to reason about solutions that satisfy certain quality attributes. In this section, we discuss some background on both variability and NFR modelling. The third part of our proposal, the CT, is explained in detail in Section 3.

2.1 Variability Modelling

Feature-oriented Domain Analysis (FODA) was the first formalisation of variability modelling and reasoning [24] as FMs. FMs are used to model the commonality and variability, and external solvers are used mainly to automatically generate the product variants. FMs are represented as a rooted tree graph – one parent, many children, composed of features as Boolean variables, and relationships (see Figure 1). Relationships among features are specified as propositional logic, including tree (e.g., And, Or) and cross-tree constraints. Consequently, it is possible to reason about FMs as a Boolean *satisfiability* (SAT) problem [7].

Several application domains, such as our IoT/EC illustrative example, require additional constructs that traditional feature models do not include. More than 45 extensions have been proposed for different needs [8], being the NVM [30] one of the most relevant for intensive variability domains. NVMs represent systems that also contain numerical features along with arithmetic cross-tree relationships (e.g., automated reasoner GreenScaler [10]).

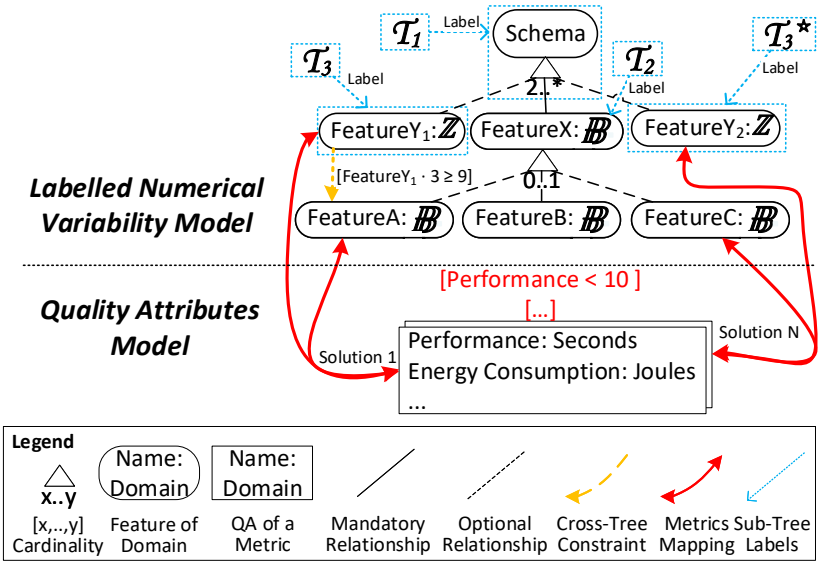

Fig. 1. Example: Relationship of NVM Solutions with QAs

NVMs consider both Boolean ($\mathbb{B}$) and discrete numerical domains such as *Integers* ($\mathbb{Z}$) [4]. As depicts in the rooted tree graph of top-Figure 1, an NVM supports the variables **and** operations of those domains together, allowing mixing $\mathbb{B}$ conditions and arithmetic (e.g., $Feature_{\mathbb{Z}}Y_1 * 3 \geq 9 \rightarrow Feature A_{\mathbb{B}}$). Another extension to FODA that we also consider in this work (see Figure 1) is the specification of the exact number of children features (i.e. feature cardinality [11]). One more extension required by variability intensive systems is the sub-tree labelling presented in top-Figure 1, which allows: (1) variability tree composition of layered NVMs [31], (2) partial instances [2], (3) cloneables like $\mathbb{T}_3$ [16], and (4) the intrinsic hierarchy between trees [19]. In summary, VMs cover the functional requirements – the actions that a system must be capable of doing. However, optimisation analyses of SPLs require NFRs - the non-behavioural aspects under which the system must operate [17] and, as previously stated in the introduction, this is not natively included as part of VMs.

2.2 Non-Functional Requirements Modelling

For this work, we define a *QA model* (QAM) as any model that specifies and hosts QAs being name-domain-metric with NFRs. Quality models [1] are a broader type of models not used in this work. QAs whose values can be quantified, such as *Performance* and *EnergyConsumption* of bottom-Figure 1, can be modelled as a set of measurements. To reason about the quality of a certain VM solution, these measurements need to be somehow linked to the variability model. In

[4] *Real* ($\mathbb{R}$), and other continuous domains, are not completely supported by SPL automated reasoners because they generate unlimited solutions.

bottom-Figure 1 there is an example of a user NFR *"Performance < 10 Seconds"* for the defined QA (e.g., performance measured as execution time). To clarify, the QAM in the example potentially encodes the total performance and energy consumption of each possible valid solution (i.e., product).

There is no consensus on how QAs measurements should be linked to features in a VM, existing two main approaches. One in which measurements are linked to individual features (i.e., each feature contributes individually to the system QA). Another, which is in line with our approach, considers that the set of measurements of a QAM should be univocally linked to a VM valid solution.

The first specific SPL solution for QAs is *Extended Variability Model* [5] already cited in Section 1, where the concept of feature in FODA is extended with attributes. Attributes have a name and a domain, and they are linked to individual features. This is useful if we consider that a feature can be assessed by a single quality measurement (e.g., an encryption code consumes 1.3 Joules). But, the quality of certain features usually cannot be assessed using a single feature. For example, to adequately assess the energy consumption of an encryption code, we need to specify several features, modelling the different key sizes, modes and paddings [27]. Therefore, we cannot model the energy consumption of an encryption code with an attribute, we need a way to link a complete solution, for example, composed by the features: AES algorithm, Mode_CBC, NoPadding, key_size = 256, to an energy measurement. Another argument is that with this approach, we can only assess the overall quality of the system as a simple direct addition of individual QAs; first, not even linear equations adjust real-world QA metrics, and second, the process of adjusting a function to a set of measurements is computationally costly, mutable, and inaccurate in average [35].

Another common approach to model QAs in SPLs is a balanced tree graph alike hierarchical activity/data models describing metrics in a top-down approach [17]. However, hierarchical trees cause extreme repetition, as each solution must be intrinsically modelled in order to connect them to their NFRs [19]. To overcome this limitation, multi-NVMs interconnected with a bunch of cross-model constraints have been proposed [22]. However: (1) hierarchical trees are useless to optimisation-type metrics (e.g., energy consumption constraining runtime metrics), and (2) so many cross-model constraints complicate the model while decreasing reasoning performance. Nonetheless, most of these solutions are not directly compatible with automated reasoners.

We use the HADAS tool [31] as a running example, where an NVM defines systems components, and an entity-relationship schema defines the QAM. Clafer [2] and the database reasoners are embedded as **Solution-to-QAs** mapping procedures, allowing hybrid automatic reasoning. Databases functionality, as querying in batches or random sampling, offers potential advantages for SPL analyses. However, the drawback of maintaining two individual but different models, the computational overhead of two co-existing reasoners, and their in-between resulting models transformations, diminish the scabalibity for very large NVMs [29]. Hence, SPL reasoning lack of a unified model that appropriately supports Boolean and numerical variability with non-functional metrics.

Every alternative contains a high degree of interlocking relationships – we are dealing with relational models. While originally, they dealt with different problems developing different methods, there are overlaps – different methods to solve the same problem. But yet, there are specific limitations of each alternative [21]. Contrarily to these approaches, we propose to abstract SPLs systems into a single relational modelling framework, where a unified semantics can jointly define seemingly dissimilar structures and the connections between them.

3 Category Theory for Software Product Lines

In this section, we give a light-weight description of Category Theory (CT) and the way we use it as a unifying modelling and reasoning framework. For a deeper introduction to CT, we refer the interested reader to [3].

Category Theory is a general mathematical theory of algebraic structures that allows the common aspects of different structures to be captured and related, while abstracting from their individual specifics. Informally speaking, a category $\mathcal{C}$ is any collection of *objects* representing spaces that can be related to each other via *arrows* (i.e., *morphisms*). Two standard 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.

Category Theory is built from the following main concepts:

- **Object**: a structured class $X \in \mathrm{Ob}(\mathcal{C})$, graphically depicted as a node $\bullet^X$.
- **Arrow**: a structure-preserving function $a \in \mathrm{Arr}(\mathcal{C})$ with source and target objects $X = src(a)$ and $Y = tgt(a)$, respectively, depicted $\overset{X}{\bullet} \xrightarrow{a} \overset{Y}{\bullet}$.
  - **Identity**: for all $X \in \mathrm{Ob}(\mathcal{C})$, we have $\overset{X}{\bullet} \xrightarrow{identity} \overset{X}{\bullet}$.
  - **Composition**: if $\overset{X}{\bullet} \xrightarrow{a_1} \overset{Y}{\bullet}$ and $\overset{Y}{\bullet} \xrightarrow{a_2} \overset{Z}{\bullet}$, then $\overset{X}{\bullet} \xrightarrow{a_1 \circ a_2} \overset{Z}{\bullet}$. It is associative, i.e., $a_1 \circ (a_2 \circ a_3) = (a_1 \circ a_2) \circ a_3$.
- **Category**: consists of $\mathrm{Ob}(\mathcal{C})$ and $\mathrm{Arr}(\mathcal{C})$. It is depicted as a directed graph.
- **Functor**: a mapping 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.

In addition, we shall need the following concepts and terminology, borrowed from a CT framework for algebraic data integration [6]:

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

In the following subsections, we illustrate intuitive examples of how to represent NVMs and QAMs as related categories. In summary, each model will be represented as a category with objects variability trees (for NVMs) and metrics sets (for QAMs), and relationships will be represented as arrows. This will allow us to generate *joint solution spaces* (i.e., SPL products with their QAs) with any automated reasoner for any type of model.

3.1 Category of Numerical Variability Models ($\mathcal{NVM}$)

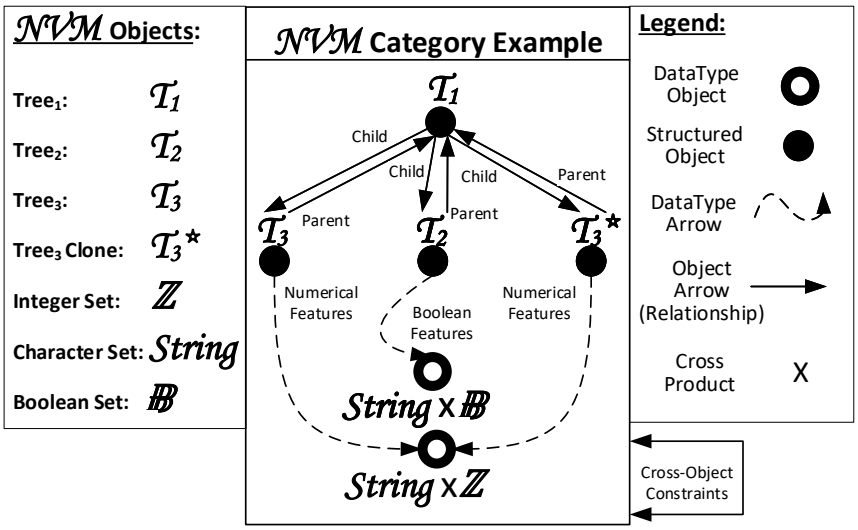

Fig. 2. $\mathcal{NVM}$ category: 4 composing variability trees ($\mathcal{T}_3$ cloned), and 2 domains

The NVM from Figure 1 is transformed into the category $\mathcal{NVM}$, depicted in Figure 2. Since the NVM is a composition of trees, Ob($\mathcal{NVM}$) is a set of four variability trees: $\mathcal{T}_1$ and $\mathcal{T}_2$ having numerical features, and $\mathcal{T}_3$ and its clone $\mathcal{T}_3*$ having Boolean features. Arr($\mathcal{NVM}$) is the set of relationships in NVMs: hierarchy (i.e., Parent/Child), cardinality, and Boolean and arithmetic cross-tree constraints. A tree trace is an $\mathcal{NVM}$ path, and an instance is populating $\mathcal{NVM}$-features with values. The basic datatype objects are programming languages library types. Here, arrow composition allows, for example, to access the Boolean value of an element (i.e., arrow) in $\mathcal{T}_1$, which was required by a parent relationship (i.e., arrow) in $\mathcal{T}_3$. These two arrows are of a different nature in NVMs, but not in $\mathcal{NVM}$. In summary, the $\mathcal{NVM}$ category is Ob($\mathcal{NVM}$) $\cup$ Arr($\mathcal{NVM}$). While the example objects are mono-type, multi-type is also supported.

3.2 Category of Quality Attributes Model ($\mathcal{QAM}$)

The QAM from Figure 1 is transformed into the category $\mathcal{QAM}$ depicted in Figure 3. As QAM is a set of valued-QAs and NFRs, Ob($\mathcal{QAM}$) consists of the *Measured* QAs ($\mathcal{M}_\mathcal{S}$) and datatype objects. Consequently, Arr($\mathcal{QAM}$) consists of NFRs and datatype arrows. Elements as *measurement* $\in \mathcal{M}_\mathcal{S}$ are of *name-domain-metric*, where the arrow is *measurement* $\xrightarrow{metric}$ $\mathcal{S}tring \times \mathbb{Z} \times \mathcal{S}tring$.

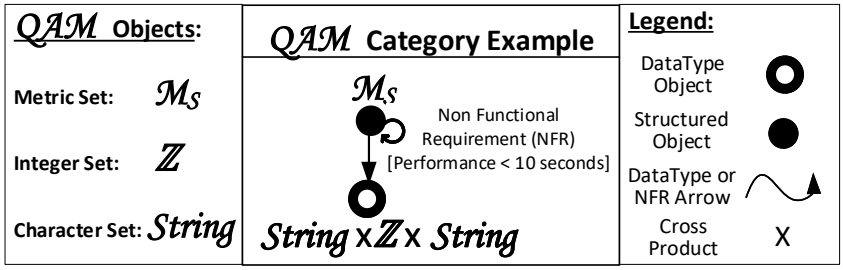

Fig. 3. $\mathcal{QAM}$ category: a metrics set object ($\mathcal{M}_S$) with a structured domain element

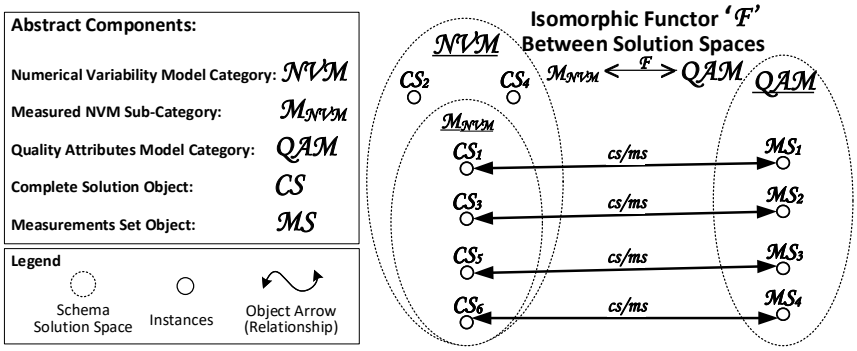

Fig. 4. Measured $\mathcal{NVM}$ and $\mathcal{QAM}$ Solution Space Isomorphism

3.3 Solution Space Categories Isomorphism

Now that we have unified the models, we shall describe how to connect a specific set of features with its specific set of QAs and values. $\mathcal{NVM}$ and $\mathcal{QAM}$ are solution-space related categories as illustrated in Figure 4. Each solution space structure is defined as a *results* object, similarly to the resulting table of a database query. In the case of $\mathcal{NVM}$, the *Complete Solution* object $\mathcal{CS}$ comprises *Sets* of elements ($\mathcal{CS}_x$) forming a unique solution according to Arr($\mathcal{NVM}$) – i.e., the satisfiable products of the SPL. Similarly, in $\mathcal{QAM}$ the object $\mathcal{MS}$ comprises the QA *Measurements Sets* $\mathcal{MS}_x$ according to Arr($\mathcal{QAM}$). The basic automated reasoners for categories are mathematical theorem provers; however, they are typically supported by other optimisation engines for specific tasks (e.g., Knuth-Bendix completion prover with a Chase searching algorithm [6]).

Some $\mathcal{CS}_x$ solutions do not correspond to $\mathcal{MS}_x$ measurements (Figure 4). The reason is that we need to consider that not every system has been measured. Hence, $\mathcal{M}_{\mathcal{NVM}}$ is the sub-category of measured $\mathcal{NVM}$, where the $\mathcal{CS}$ object has a bijective (i.e., one-to-one) arrow *cs/ml* to the $\mathcal{ML}$ object of $\mathcal{QAM}$. Consequently, there is an isomorphic functor[5] [20] between $\mathcal{M}_{\mathcal{NVM}}$ and $\mathcal{QAM}$.

[5] A categories isomorphism is a one-to-one mapping between their sets of objects.

4 Validation and Discussion

To validate our framework, we deploy a CT prototype of a running SPL tool[6] – the NVM and QAM optimisation assistant for Edge Computing HADAS [31].Edge devices were defined in a composed Clafer NVM (left-side Figure 5), with two main trees: Hardware and Software. The last is composed of four trees: *Operating System* (OS), *Programming Language* (PL), *Operation* and *Context*. Again, the latter is composed of *Libraries* and *Numerical Parameters* trees. All the trees have Boolean features, besides *Numerical Parameters*, which only contains Integer features (e.g., Encryption_Key: 64 bytes [29]). HADAS QAM is a relational database that links NVM solution-tree leaves with QAM dynamic identifiers.

4.1 $\mathcal{HADAS}$ category

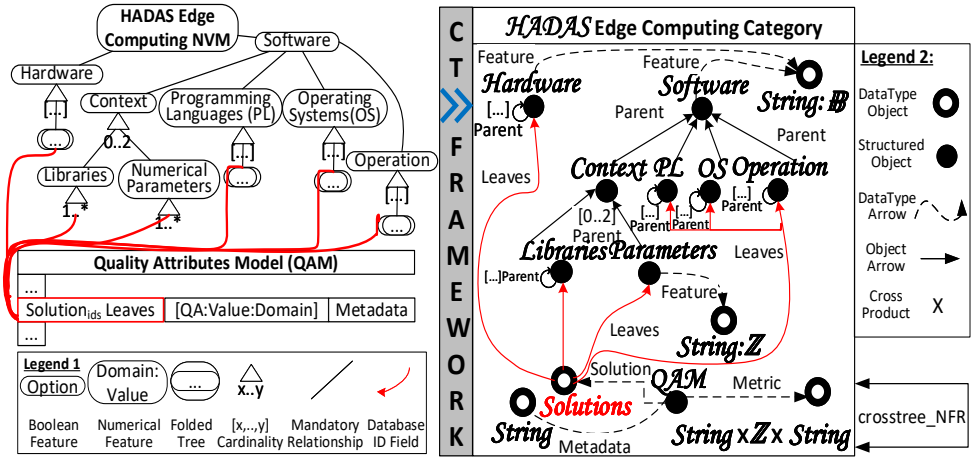

Fig. 5. Transformation of HADAS NVM and QAM in a single $\mathcal{NVM}$ category

In Figure 5 we can see on the left the HADAS base NVM and QAM structures, and on the right its unified category $\mathcal{HADAS}$ by means of the framework presented in Section 3. Our framework is as flexible as CT; existing models can be transformed into categories differently and yet perform equally. For example, an object could be modelled as a category with a single object and vice-versa. Our philosophy in this proof-of-concept is to keep the category simple; hence, we applied this example combining $\mathcal{NVM}$ and the single object category $\mathcal{QAM}$ into $\mathcal{HADAS}$, where the technical implication is switching the categories functor by an objects arrow. In summary, data-types, NVM trees and QAM are 12 $\mathcal{HADAS}$ objects, and variability trees relationships, cross-tree constraints and NFRs are a minimum of 6 arrows. $\mathcal{HADAS}$ consists of the following components:

[6] HADAS web-services: `https://hadas.caosd.lcc.uma.es/`

- Ob($\mathcal{HADAS}$) $\ni$ $\mathcal{S}tring$, $\mathcal{S}tring \times \mathbb{B}$, $\mathcal{S}tring \times \mathbb{Z} \times \mathcal{S}tring$, $\mathcal{H}ardware$, $\mathcal{S}oftware$, $\mathcal{PL}$, $\mathcal{OS}$, $\mathcal{O}peration$, $\mathcal{C}ontext$, $\mathcal{L}ibraries$, $\mathcal{P}arameters$, $\mathcal{S}olutions$.
- Arr($\mathcal{HADAS}$) $\ni$ *feature*, *metric*, *parent*, *cardinality* (e.g., [0..2]), *metadata*, *crosstree_NFR*, *solution*, *leaves*. In occasions *parent* is the *identity* (Figure 5).
- Elements: based on Arr($\mathcal{HADAS}$), there are Boolean and integer *features*, QA *metrics* with format *name-domain-metric*, QA *metadata*, and *solution* the set of object features leaves to QAs (i.e., $\mathcal{HADAS}$ solution space).

4.2 Optimal deployment

The next step in this proof-of-concept is to instantiate (i.e., populate) $\mathcal{HADAS}$, to later generate the solution space (e.g. IoT/EC deployments), and optimise its QAs. EC and IoT systems require fast real-time processing of random amounts of data and have relatively strict NFRs on the performance and energy consumption [34]. Hence, we propose to turn into a category the model shown in Figure 6 on the left, aiming to gain insights into which features and solutions are affecting those QAs in transmitting and/or compressing operations. The NVM contains 28 Boolean features and two numerical features, while the QAM contains two QAs – performance in Seconds and energy rate in milliWatts. Operations are partial configurable benchmarks of the Phoronix Test Suite[7].

Having a clear picture of the category base model in Figure 5, we need to program and deploy it. While there are libraries aiming to add CT support to SPL reasoners (e.g., Conal Elliot libraries for Z3 [14]), the only production-ready *Integrated Development Environment* (IDE) is the *Categorical Query Language* (CQL) IDE: an open-source software, commercialised by Conexus AI[8]. It is a canonical functional IDE that generates CT graphs as the presented figures.

On the right of Figure 6, there is a partial code-snapshot; the CQL model can be downloaded from the HADAS server[9]. There one can find the 30 NVM features and the 2 QAs distributed in the $\mathcal{HADAS}$ objects shown on the right side of Figure 6. We did not include cross-tree constraints in the graph due to extension limitations; however, they are arrows in $\mathcal{HADAS}$.

4.3 Results and Discussion

CQL IDE reasoning is automatically performed with a combination of different algorithms – the key of our performance. We used them as such: automated theorem prover with Knuth-Bendix completion [28] for logic and equations, and hashing, balanced trees and chasing for data-type and cross-object arrows.

We have obtained 162 valid solutions with their respective 324 measurements in 0.1 Seconds. If we reduce the category, the runtime is still 0.1 Seconds. Extending the category as a supra-category formed by a self-cross-product 3 times results in 0.2 Seconds. Running CQL IDE on another computer did not change

[7] Phoronix Test Suite details: `https://openbenchmarking.org/tests/pts`
[8] CQL IDE main website: `https://www.categoricaldata.net/`
[9] HADAS CQL CT model: `https://hadas.caosd.lcc.uma.es/ctprototype.cql`

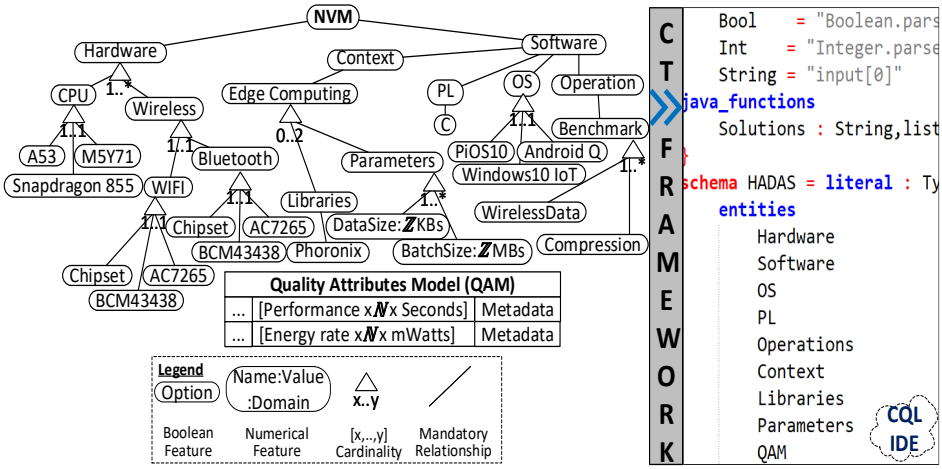

Fig. 6. Transformation of HADAS for Edge Computing in CQL IDE $\mathcal{HADAS}$ category

the runtimes. This suggests that CQL IDE scales linearly, and that the minimum runtime is 0.1 Seconds independently of the computer, probably due to being a Java application running on a Java virtual machine.

Optimisation arrows are a step further from the solution space. Maximising performance or minimising energy rate increases the reasoning runtime by 0.1 Seconds, independently of the solution space size. However, we expect linear increments for larger models (i.e., linear scalability). Regarding the interaction of features with regards to the QAs in our EC case study, the main insights are:

- Compressing/uncompressing while sending/receiving data improves the runtimes for large batches of data, but for small ones, it is the opposite independently of the original data size. In any case, compressing increased the energy rate – more Joules per Second.
- The more powerful the CPU is, the lesser is the compressing time, and the higher the energy rate; the maximum energy rate of *Snapdragon 855* was 3.4 Watts, of *A53* was 3.7 Watts, and of *M5Y71* was 11.8 Watts. In the case of communication without compression, CPUs barely affected QAs.
- Communication peripherals affected equally the QAs. WiFi and Bluetooth channels performed equally for small batches of data. WiFi has tends to be faster above 300 MegaBytes, while the Bluetooth energy-rate is substantially lower (with an average of 0.5 Watts) than these 300 MegaBytes.

As **internal validity**, we tested our proposal by first transforming an SPL tool into a category, and second, by modelling an EC case study in a category. Additionally, we implemented that category in CQL IDE, performing reasoning to generate the solution space, and also performing an optimal search to obtain quality insights from the EC category. To mitigate scalability assumptions, we ran CQL IDE with different solution spaces in different computers.

As **external validity**, we identified two threats. First, we have not tested our approach on large models with other IDEs, since our aim was a proof-of-

concept. Second, as it is the first CT NVM/QAM framework to the best of our knowledge, we cannot compare it with other CT alternatives.

5 Related Work

Table 1. Characteristics of abstraction alternatives for a unified NVM/QAM

→**Model** ↓***Entity***	**NVM**	**Category Theory**	**Set Theory**	**FODA**	**Codd Algebra**	**HOL**	**Arithmetic**
Structured Model	Labelled NVM	Category	Finite Set	Labelled FM	Data Schema	Logic Formula	Plane
Entities	Sub-tree, Feature, Solution	Sub-Category, Object, Instance	Sub-Set, Element	Sub-tree, Feature, Configu-ration	*Table ,Cell*	Partial Formula, Variable	Sub-System, Equation ,Variable
Boolean Type	$\mathbb{B}$ Feature	$\mathbb{B}$	$\mathbb{B}$	Feature	$\mathbb{B}$	$\mathbb{B}$	*Pseudo-*$\mathbb{B}$*: [0,1]*
Numerical Type	$\mathbb{N}$, $\mathbb{Z}$, $\mathbb{R}$ Feature	$\mathbb{N}$, $\mathbb{Z}$, $\mathbb{R}$	$\mathbb{N}$, $\mathbb{Z}$ *Finite Sets*	*Un-supported*	$\mathbb{N}$, $\mathbb{Z}$, $\mathbb{R}$	*Un-supported*	$\mathbb{N}$, $\mathbb{Z}$, $\mathbb{R}$
Requi-rements	Cross-tree Equation	Categories Functor	Predicate	Cross-tree Constraint	σ Constraint	Formula	Equation
Selection	Assert	Δ	$\in$	Require	$\pi_{[\text{Column}]}$	True	$=$
Exclusion	Not	$-\Delta$	$\notin$	Exclude	$-\pi_{[\text{Column}]}$	False	$\neq$
Con-nectives	&&, |||, [x..y], $\Rightarrow$, $\equiv$	$\sum_{[\text{Functor}]}$ $\prod_{[\text{Functor}]}$, X	$\wedge$, $\vee$, $\oplus$, If, $\Rightarrow$, $\equiv$	And, Or, xOR, $\Rightarrow$, $\equiv$	Foreign Key, $\bigcap$, $\bigcup$, $\biguplus$, $\text{Joins}_{[X]}$	$\wedge$, $\vee$, $\oplus$, If, $\Rightarrow$, $\equiv$ $\forall$, $\exists$, !	Equation Systems
Equalities	$=, \neq, >, \geq, <, \leq$	$=, \neq, >, \geq, <, \leq$	$=, \neq$	$=, \neq$	$=, \neq, >, \geq, <, \leq$	$=, \neq$	$=, \neq, >, \geq, <, \leq$
Mathe-matics	+, -, *, $\div$, %...	+, -, *, $\div$, %...	*Pre/Suc-cessor*	*Un-supported*	*Un-supported*	*Un-supported*	+, -, *, $\div$, %...

Having already presented the relevant publications for the foundations of this paper, we now discuss further related work. Firstly, while we have discussed the advantages of CT, one could argue that more simple structures could be used instead to unify NVMs with QAMs. In Table 1 there is a summary of the alternatives, where we highlight first the needs of NVMs, and second what CT provided as a reference. Whether we are talking about NVMs with or without QAMs, we need a complete Boolean and numerical domain. *FODA*, the first VM formalisation [24], has already been discussed and discarded due to its lack of support for numerical features and constraints necessary in EC analyses. In fact, as identified in Table 1, most of the alternatives lack numerical support. One of them is *Set Theory* (ST), which, similarly to CT, is a branch of mathematical logic that studies sets, which informally are collections of objects [15]. ST lacks

support for numerical equations, inequalities, and infinite data-types. Similarly, *Higher-Order Logic*(HOL) deals just with declarative propositions, predicates and quantification (e.g., $\forall x$) [26]. *Codd Theory* is the first and only formalisation of relational algebra, which uses algebraic structures with well-founded semantics for modelling data and defining queries on it. While databases support a wide range of numerical components as datatypes, counting, grouping, arithmetic, etc., they are programming workarounds outside of Codd Theory. In other words, it is not yet clear that Codd relational algebra should be extended above a pure Boolean domain [25]. Pseudo-Boolean (i.e. [0,1]) reasoners are based on *Arithmetic* [25] – the study of numbers and their operations. While they are promising, if not considering the complexity and overload of model-transforming HOL, their performance in current SAT competitions is often quite poor [13]. It should be pointed out that all of the above-mentioned theories (ST, HOL, Codd Algebra) are well-formalised categories in CT.

A computational design framework based solely on objects and arrows is proposed in [4], where Model Driven Engineering meets (Boolean) SPLs. This approach was extended with explicit use of CT in [36], where VMs and Domain Models are unified. In Clafer SPL suite, VMs are modelled as abstract classes, literally an idea borrowed from CT [2]. A generic CT approach for different data domains integration is formalised in [6], where as a case study entity-relational models (i.e., database models) are transformed into a category in which tables are objects, columns are elements, and foreign keys are arrows.

6 Conclusions and Future Work

In this paper, we identify the lack of automated tools to model and optimise SPLs defined as an NVM related to sets of QAs with values. To address this, we define a unified model supporting: (1) Boolean and numerical domains in the form of features and their relationships, and (2) a map between the solution spaces of NVMs and QAMs. For that, we propose a CT framework with two categories. The first one is $\mathcal{NVM}$ where variability trees and data-types are objects, and hierarchical and cross-tree constraints are arrows. The second one is $\mathcal{QAM}$ where the sets of QAs and their data-types are objects, and NFRs are arrows. Finally, we establish a functorial relationship between measured products of $\mathcal{NVM}$ with QAs sets of $\mathcal{QAM}$. As a proof-of-concept, we transformed the SPL HADAS into the category $\mathcal{HADAS}$. Then, we have implemented and deployed it in the CQL IDE and performed a brief EC case study using a combination of theorem provers and database algorithms as automated reasoners. As future work, we plan to improve the framework to support other proposed extended functionalities of NVMs, as well as integrate quality models. Currently, we are in the process of evaluating this approach with large SPLs.

Acknowledgements 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. Al-Qutaish, R.E.: Quality models in software engineering literature: an analytical and comparative study. Journal of American Science **6**(3), 166–175 (2010)
2. Bak, K., Diskin, Z., Antkiewicz, M., Czarnecki, K., Wasowski, A.: Clafer: unifying class and feature modeling. Software & Systems Modeling **15**(3), 811–845 (2016)
3. Barr, M., Wells, C.: Category theory for computing science. Prentice Hall (1990)
4. Batory, D., Azanza, M., Saraiva, J.: The objects and arrows of computational design. In: International Conference on Model Driven Engineering Languages and Systems. pp. 1–20. Springer (2008)
5. Benavides, D., Trinidad, P., Ruiz-Cortés, A.: Automated reasoning on feature models. In: Pastor, O., Falcão e Cunha, J. (eds.) Advanced Information Systems Engineering. pp. 491–503. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)
6. Brown, K.S., Spivak, D.I., Wisnesky, R.: Categorical data integration for computational science. Computational Materials Science **164**, 127–132 (2019)
7. Budiardjo, E.K., Zamzami, E.M., et al.: Feature modeling and variability modeling syntactic notation comparison and mapping. Computer and Communications ('14)
8. Chen, L., Ali Babar, M., Ali, N.: Variability management in software product lines: A systematic review. In: Proceedings of the 13th International Software Product Line Conference. p. 81–90. SPLC '09, Carnegie Mellon University, USA (2009)
9. Chohan, A.Z., Bibi, A., Motla, Y.H.: Optimized software product line architecture and feature modeling in improvement of spl. In: 2017 International Conference on Frontiers of Information Technology (FIT). pp. 167–172. IEEE (2017)
10. Chowdhury, S., Borle, S., Romansky, S., Hindle, A.: Greenscaler: training software energy models with automatic test generation. Empirical Software Engineering **24**(4), 1649–1692 (2019)
11. Czarnecki, K., Helsen, S., Eisenecker, U.: Formalizing cardinality-based feature models and their specialization. Software process **10**(1), 7–29 (2005)
12. De Moura, L., Bjørner, N.: Z3: An efficient smt solver. In: Int. conference on Tools and Algorithms for the Construction and Analysis of Systems. Springer (2008)
13. Elffers, J., Giráldez-Cru, J., Nordström, J., Vinyals, M.: Using combinatorial benchmarks to probe the reasoning power of pseudo-boolean solvers. In: Int. Conference on Theory and Applications of Satisfiability Testing. Springer (2018)
14. Elliott, C.: Compiling to categories. Proceedings of the ACM on Programming Languages **1**(ICFP), 1–27 (2017)
15. Fraenkel, A.A., Bar-Hillel, Y., Levy, A.: Foundations of set theory. Elsevier (1973)
16. Gamez, N., Fuentes, L.: Software product line evolution with cardinality-based feature models. In: Int. Conference on Software Reuse. pp. 102–118. Springer (2011)
17. Glinz, M.: On non-functional requirements. In: 15th IEEE International Requirements Engineering Conference (RE 2007). pp. 21–26. IEEE (2007)
18. González-Huerta, J., Insfran, E., Abrahão, S., McGregor, J.D.: Non-functional requirements in model-driven software product line engineering. In: Proceedings of the Fourth International Workshop on Nonfunctional System Properties in Domain Specific Modeling Languages. NFPinDSML '12, NY, USA (2012)
19. Gurov, D., Østvold, B.M., Schaefer, I.: A hierarchical variability model for software product lines. In: International Symposium On Leveraging Applications of Formal Methods, Verification and Validation. pp. 181–199. Springer (2011)

20. Gurrola-Ramos, L., Macías, S., Macías-Díaz, J.: On the isomorphism of injective objects in grothendieck categories. Quaestiones Mathematicae **40**(5) (2017)
21. Hellendoorn, V.J., Sutton, C., Singh, R., Maniatis, P., Bieber, D.: Global relational models of source code. In: Int. Conference on Learning Representations (2019)
22. Horcas, J.M., Pinto, M., Fuentes, L.: An automatic process for weaving functional quality attributes using a software product line approach. Journal of Systems and Software **112**, 78 – 95 (2016)
23. Jussien, N., Rochart, G., Lorca, X.: Choco: an open source java constraint programming library. In: HAL Archives Ouvertes (2008)
24. Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S.: Feature-oriented domain analysis (foda) feasibility study. Tech. rep., Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst (1990)
25. Kızıltoprak, A., Köse, N.Y.: Relational thinking: The bridge between arithmetic and algebra. International Journal of Elementary Education **10**(1), 131–145 (2017)
26. Lambek, J., Scott, P.J.: Introduction to higher-order categorical logic, vol. 7. Cambridge University Press (1988)
27. Montenegro, J.A., Pinto, M., Fuentes, L.: What do software developers need to know to build secure energy-efficient android applications? IEEE Access (2018)
28. Müller, J.: Theopogles—a theorem prover based on first-order polynomials and a special knuth-bendix procedure. In: GWAI-87 11th German Workshop on Artifical Intelligence. pp. 241–250. Springer (1987)
29. Munoz, D.J., Montenegro, J.A., Pinto, M., Fuentes, L.: Energy-aware environments for the development of green applications for cyber–physical systems. Future Generation Computer Systems **91**, 536 – 554 (2019)
30. Munoz, D.J., Oh, J., Pinto, M., Fuentes, L., Batory, D.: 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. p. 289–301. Association for Computing Machinery, NY, USA (2019)
31. Munoz, D.J., Pinto, M., Fuentes, L.: Hadas: analysing quality attributes of software configurations. In: Proceedings of the 23rd International Systems and Software Product Line Conference-Volume B. pp. 13–16 (2019)
32. Olaechea, R., Stewart, S., Czarnecki, K., Rayside, D.: 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. pp. 1–6 (2012)
33. Pohl, K., Böckle, G., van Der Linden, F.J.: Software product line engineering: foundations, principles and techniques. Springer Science & Business Media (2005)
34. Ren, J., Wang, H., Hou, T., Zheng, S., Tang, C.: Federated learning-based computation offloading optimization in edge computing-supported internet of things. IEEE Access **7**, 69194–69201 (2019)
35. Siegmund, N., Rosenmüller, M., Kuhlemann, M., Kästner, C., Apel, S., Saake, G.: Spl conqueror: Toward optimization of non-functional properties in software product lines. Software Quality Journal **20**(3-4), 487–517 (2012)
36. Taentzer, G., Salay, R., Strüber, D., Chechik, M.: Transformations of software product lines: A generalizing framework based on category theory. In: 20th Int. Conference on Model Driven Engineering Languages and Systems (MODELS) (2017)