HAL
open science

Multifaceted Automated Analyses for Variability-Intensive Embedded Systems

Sami Lazreg, Maxime Cordy, Philippe Collet, Patrick Heymans, Sébastien Mosser

To cite this version:

Sami Lazreg, Maxime Cordy, Philippe Collet, Patrick Heymans, Sébastien Mosser. Multifaceted Automated Analyses for Variability-Intensive Embedded Systems. 41st ACM/IEEE International Conference on Software Engineering, May 2019, Montréal, Canada. hal-02061251

HAL Id: hal-02061251

https://hal.science/hal-02061251

Submitted on 8 Mar 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Multifaceted Automated Analyses for Variability-Intensive Embedded Systems

Sami Lazreg
Visteon Electronics
Université Côte d'Azur
CNRS, I3S, France

Maxime Cordy*
SnT, University of Luxembourg
Luxembourg

Philippe Collet
Université Côte d'Azur
CNRS, I3S, France

Patrick Heymans
University of Namur
Belgium

Sébastien Mosser
Université Côte d'Azur
CNRS, I3S, France

***Abstract*—Embedded systems, like those found in the automotive domain, must comply with stringent functional and non-functional requirements. To fulfil these requirements, engineers are confronted with a plethora of design alternatives both at the software and hardware level, out of which they must select the optimal solution wrt. possibly-antagonistic quality attributes (e.g. cost of manufacturing vs. speed of execution). We propose a model-driven framework to assist engineers in this choice. It captures high-level specifications of the system in the form of variable dataflows and configurable hardware platforms. A mapping algorithm then derives the design space, i.e. the set of compatible pairs of application and platform variants, and a variability-aware executable model, which encodes the functional and non-functional behaviour of all viable system variants. Novel verification algorithms then pinpoint the optimal system variants efficiently. The benefits of our approach are evaluated through a real-world case study from the automotive industry.**

I. Introduction

In many embedded systems, requirements engineering and design activities are tightly intertwined and involve complex multi-criteria decision-making over various concerns. As an example, let us consider an infotainment feature in an automotive system. Specifying such a feature typically entails defining a set of functional and non-functional (aka. quality) requirements. Functional requirements define, for example, not only what content should be displayed through the Human Machine Interface (HMI), but also constraints imposed by the hardware , like not exceeding the available memory. Typical examples of non-functional requirements constrain manufacturing costs or execution time. A notoriously difficult problem is to establish whether such a set of requirements is feasible and what is the best design to implement it.

Design options are not only constrained by the requirements but also by the existing software and hardware architectures. In our case, an HMI-rendering automotive system consists of (1) a data processing *application* (i.e., data-flow oriented embedded software) and (2) a resource-constrained hardware *platform* (i.e., heterogeneous hardware components like non-programmable processors and data storage units). The application and the platform are, however, not completely fixed as they have variation points. There are three main sources of extensive variability. First, at the application level, multiple data-flow variants can achieve the functional requirements, differing in, e.g., the size of the flowing data chunks, the ordering of the operation tasks, or the choice between alternative, functionally-equivalent tasks. Second, there exists a diversity of configurable hardware platforms which can differ, e.g., in memory capacities and processing pipelines. Third, there are various ways of mapping and deploying a given application on a specific platform, e.g., choose a processor to perform a given task or select a memory unit to store a given data.

This threefold variability is typical in automotive and many other kinds of embedded systems [1]. Unfortunately, it leads to a high number of variants (1,548,288 in this particular case), each of which represents a specific *system design alternative* (or *design* for short), that is, a specific mapping of a specific application variant to a specific platform variant. Among these design alternatives, not all are able to realize the functional requirements to the same extent, and the same holds for the non-functional requirements. Given the sheer number of variants, a systematic consideration of all design alternatives is unfeasible for the software and system engineers whereas the high level of competition in industry puts a high pressure on them to deliver optimal solutions and do so timely [2]. Efficient automations therefore appear as a necessity.

Examples of questions the engineers need to quickly answer are: *Can the specified HMI be properly rendered on platform X? Which feasible designs can be built with a budget of Y? Which feasible designs can execute in less than Z time? Which feasible designs, with a rendering quality higher than P and a manufacturing cost lower than Q, exhibit the fastest execution time? Which feasible designs reach the best tradeoff between rendering quality, manufacturing cost and execution time?* Answering those questions not only requires a way to deal with the variability-induced combinatorial explosion (see previous paragraph), but also a way to reason simultaneously about different types of concerns: feasibility/satisfiability *and* optimality; functional *and* non-functional requirements; the structure *and* the behaviour of the system. Although significant progress was made in the recent years to automate reasoning on variability-intensive systems, existing solutions only address specific facets of the problem in isolation. As revealed by our experiments, partial solutions give suboptimal designs, while complete but non-variability-aware solutions do not scale. Hence the need for multifaceted, variability-aware analyses capable of answering all the above questions.

* This work was done while Maxime Cordy was part of the U. of Namur.

Without such solutions, engineers mostly resort to intuition and experience. Theoretical analyses can be made but are time-consuming and often turn out largely suboptimal, if not completely wrong. Simulators are sometimes provided by platform suppliers but analyzing all system variants requires simulations for all of them, which is unrealistic. Quick prototyping may occur when the platform is finally supplied but it is then too late to backtrack if the wrong platform was picked. In the end, current practices are deemed very unsatisfactory.

These observations were made by our partner Visteon Electronics, an international leader in automotive systems, and are corroborated by surveys such as [3]. They formed the motivation for the industry-academia partnership which led to this research effort. In this paper, we propose an approach that combines and extends existing research results in order to provide the first tooled framework able to solve the multifaceted problem described above. Our contributions are:

1) *modelling languages* based on Y-chart [4], [5] to capture functional and non-functional specifications of variability-intensive embedded systems that can vary at both the application and platform levels;
2) *mapping algorithms* to derive, from the application and platform models, the resulting design space (i.e. all design alternatives) while capturing all the structural, behavioural, functional and non-functional variations;
3) the first variability-aware *model-checking algorithm* to optimize multiple behaviour-dependent quality attributes across all variants;
4) *an integrated tool chain* to evaluate the functional feasibility and the non-functional satisfiability and optimality of the whole design space at once;
5) *qualitative and quantitative evaluations* of the approach based on a real system developed by Visteon. The qualitative evaluation shows that functional and non-functional requirements were properly captured by our framework, and optimal system designs were correctly identified. The quantitative evaluations assess the scalability of our approach and give us confidence that our framework is applicable to the majority of Visteon's systems and similar systems developed elsewhere in industry.

Although its motivation originates from an industrial partnership, our contribution is not domain specific. Our modelling method covers a large class of variability-intensive data-flow-oriented systems with quality attributes. Our model-checking algorithm is language-independent and can be applied to a broader range of variability-intensive systems.

II. Industrial Case Study

This research originates from a collaboration with Visteon Electronics, a leading company developing solutions for the automotive industry such as instrument clusters, infotainment and connected vehicles. In this section, we introduce a simplified exerpt of the system we use in our evaluations (see Sec. VIII) to further illustrate and justify our approach.

An *instrument cluster* generally consists of a speedometer and other instruments which, unlike traditional analog gauges,

Fig. 1: Our instrument cluster application case study

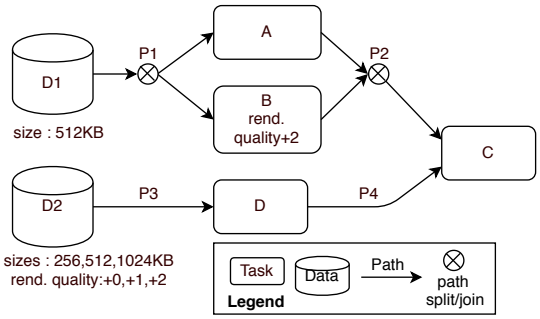

Fig. 2: A variable data-flow specification

appear on an electronic visual display (see Fig. 1). By applying various data-flow image processing effects (e.g., 3D gauges, 3D view of the car), it improves the driving experience.

To achieve economies of scale, such systems are often built and sold to car manufacturers as product lines, which have to meet and optimize the manufacturers' variable requirements for an entire range of cars. To be competitive, they are highly constrained on quality and cost. As such, one must know as early as possible in the development whether the expected HMI can be rendered properly on a candidate platform.

At industrial scale, the threefold variability introduced in Sec.I (i.e. from application, platform and mapping) prevents any product-based exhaustive feasibility checking, let alone exhaustive reasoning/optimization on *quality attributes* (e.g. cost, rendering quality, runtime). Yet, some separation of concerns is intrinsically possible as data-flows can be abstracted from the implementation by domain experts (i.e. rendering engineers). Platforms are already specified by hardware experts to organize competitive tendering on hardware providers.

Fig. 2 shows an example of data-flow specification with quality attributes. The different processing flows that meet the HMI functional requirements are captured by a variable data-flow. Images are processed by graphical tasks. Image `D1` can be processed by tasks `A` or `B`. `D2` has three different possible resolutions and is processed by task `D`. The images produced by `A` (or `B`) and `D` are then processed by task `C`, which delivers the final result. Task and image resolution impacts the HMI rendering quality. In our case, performing `B` instead of `A` significantly improves the rendering quality. Also, as the resolution of `D2` is increased, the overall quality raises as well.

At the platform level (Fig. 3), image processing functions `A,B,C,D` are provided by a non-programmable Graphical Processing Unit (`GPU`) and an advanced Display Controller Unit (`DCU`). Within `DCU`, there are three functions (`A`, `C` and `D`)

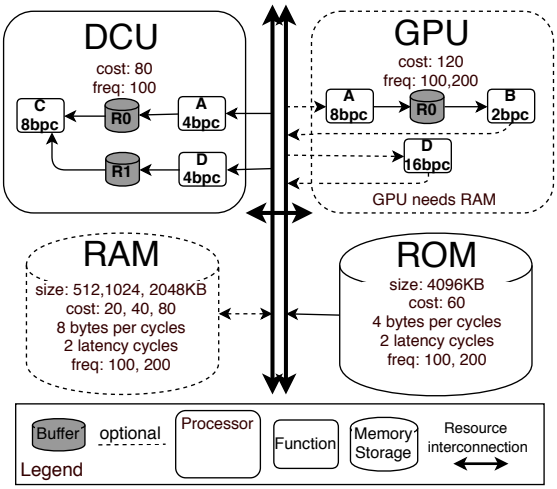

Fig. 3: Platform Specification Input Model

and two buffers (`R0` and `R1`). Directed edges denote the flows followed by data that transit through the processing pipeline of processors, while functions may be applied or not. The platform also includes data storage on `RAM` and `ROM`. `GPU` and `RAM` are optional as everything can be stored in `ROM` and then rendered directly to the screen by `DCU`. Yet, they improve the runtime efficiency for a higher manufacturing cost. There is a presence condition between `RAM` and `GPU` as `RAM` acts as a dedicated cache memory for `GPU`, while `RAM` can be used without `GPU`, e.g., to store more data or larger images. `RAM` comes in three alternative capacities (at different costs). Similarly, `RAM`, `GPU` and `ROM` have two possible frequencies. Frequency acts as a scale factor for data processing/transfer bandwidth for processors and memories.

Our overall goal is to assist engineers in determining which variants (i.e. alternative designs consisting of a data-flow variant deployed on a platform configuration) of the instrument cluster can satisfy the imposed functional (FC) and non-functional (NF) requirements.

To ensure the executabilty of the application on the platform, a first FC requirement states: "Any data required by a task must be stored in a memory storage accessible to the processor that processes the task". Its satisfaction depends only on the *structure* of the design, e.g. which tasks must be processed, which tasks exchange data with each other, which memory storage is accessible by the processors, and how tasks (resp. data) are mapped onto processors (resp. memory storage). Another FC requirement states that the execution of the application on the platform must eventually terminate. This not only depends on the structure of the design but also on its runtime *behaviour*, as bad scheduling of tasks and data transfers may cause deadlocks. Checking the satisfaction of this requirement by a given design is more complicated, as it requires analysing many (if not all) its executions.

In addition to FC requirements, the design must also meet NF requirements. These commonly include a maximal manufacturing cost, a minimal rendering quality, and a maximal execution time (i.e. time to render graphics on the visual display). Manufacturing cost and rendering quality are quality attributes which depend only on the *structure* of the design (e.g. size of input data, choice between alternative tasks, components of the platform). Execution time, however, depends on both structure (e.g. processor frequency) and behaviour (e.g. scheduling of tasks and memory access operations).

Those requirements are not enough, though, as market competition forces engineers to deliver the best system to each specific customer. Among the variants, they must thus find those offering the best trade-off between the quality attributes.

Achieving our goal thus requires solving the problem of efficiently identifying the variants that:

1) *Satisfy the FC requirements*. We further decompose this sub-problem in two challenges: (a) checking the FC requirements that depend only on the structure of the design (challenge **FCS**) and (b) checking FC requirements that depend also on its behaviour (**FCB**).
2) *Satisfy the NF requirements*, that is, checking: (a) those that depend only on the structure (**NFS**) and (b) those that also depend on the behaviour (**NFB**).
3) *Optimize the trade-off between the quality attributes* (**NFOB**). This challenge requires considering all quality attributes. Optimizing only those that depend on the structure (**NFOS**) is easier, but leads to suboptimal solutions (as revealed by our evaluations).

We must solve all those challenges to give engineers the means of making appropriate design decisions at an early stage.

III. State of the Art and Related Work

Many approaches were proposed to tackle parts of the above challenges. First, research in variability modeling is black-box oriented and focused on structural aspects. It attempts to assess [6], [7] or efficiently predict [8]–[13] non-functional properties of the whole product line, thus tackling both FCS and NFS. Extensions of these methods with multi-objective optimization [14]–[19] allow to find optimal variants wrt. structural quality attributes, thereby solving NFOS. However, those approaches lack reasoning support on the system behaviour and are thus unable to, e.g., search for optimal schedulings.

Behavioural analyses of variable systems were addressed by variability-aware model checking against functional requirements (challenge FCB) [20]–[23] or one particular non-functional aspect (e.g. real time [24]–[26], reliability [27], [28], income [19], quality of service [29]). Resource-optimal execution [30]–[33] and worst-case execution time [34] are still determined product by product. Also, contrary to our framework, these approaches are unable to automatically map variable workflow specifications on configurable platform descriptions in order to infer a system design space. Therefore, applying them obliges to manually model and assess all possible systems designs. They thus inefficiently tackle NFOB. Even when both levels are captured in software/hardware product lines with dependencies and constraints [35]–[40], the expressiveness is also insufficient, because either behaviour is not considered or only functional requirements are checked.

Some system design frameworks [4], [41]–[48], model, assess and optimize quality attributes (NFB and NFOB), but do not capture nor manage all variability dimensions. Specific techniques attempt to efficiently handle either platform variability [49]–[51], application variability [52]–[55] or deployment variability [56], but they are limited to one dimension at a time and cannot reason on the whole problem.

Other approaches [57]–[59] tackle both functional and platform variabilities by focusing on an optimal platform configuration for a multi-variant application. Contrarily to our solution, they do not find a design (i.e., a mapping of a functional variant onto a platform configuration) that ensures an optimal execution. Our recent work [23] can capture and reason on variability at all three levels. It is, however, limited to functional requirements (challenges FCS and FCB).

IV. Overview of the Framework

Our approach follows the model-driven Y-Chart process [4], [5] which explicitly separates application and platform. This allows for the modular specification of, and the reasoning on, the different parts of the system. As shown in Fig. 4, our framework involves several models and processes. On the left, the inputs are application and platform models such as those of Fig. 2 and Fig. 3. Domain experts are expected to model the application as an extended concurrent data-flow model with quality attributes. The model contains the classic structure and behaviour of the data-flow (data, task, edge) and its variability. Additionally, platform experts provide the platform specification as a templated concurrent component-based system, which also captures platform variability.

First process (detailed in Sec. V): from the input models the framework generates a *variability-intensive design space* that captures all valid deployment mappings of the variable application onto the configurable platform. A mapping basically allocates tasks to processors and data to memory components.

Second process (detailed in Sec. VI): from the design space, we generate representations that allow for reasoning on the structure and the behaviour of all designs. In addition to the input models, experts define the NF requirements as constraints and a cost function representing trade-offs between quality attributes. To relate NF requirements with the design space, we rely on feature models with quality attributes [10], [14] for the structural part, and on featured transition systems [22] with added weights [60] for the behavioural part.

Third process (detailed in Sec. VII): we reuse feature-model reasoning and apply our new model-checking algorithms to identify the designs that meet the requirements (FCS, FCB, NFS and NFB) and are optimal (NFOB). Additionally, we can provide the execution traces that optimize the quality attributes while satisfying the requirements. Such a trace shows not only the designs able to execute it but also the behaviour it exhibits (i.e., how the application tasks are executed and scheduled onto the platform), thereby helping the upcoming engineering of the designs.

V. Applications, Platforms and Mappings

To model the application, the platform and their quality attributes, we extend our former modelling framework [23] to support NF requirements, as it is limited to checking designs that satisfy the FC requirements. Thus, we define formal models for variable dataflows and configurable platforms.

Definition 1: A variable data-flow graph is a tuple $VDG = (N, P, E, \Psi, \chi_d)$ where $N = T \cup D$ is a set of nodes (T are tasks and D are data); P is a set of communication paths by which data flows between producers (tasks and data) and consumers (tasks); $E \subseteq N \times P \times T$ is the set of flow processing between producers and consumers via available paths; Ψ is a set of attributes; $\chi_d : N \rightarrow (\Psi \rightarrow \bigcup_{\psi \in \Psi} \nu(\psi)) \rightarrow \{true, false\}$ is a function that associates a node n to the set of values that (all or a subset of) the attributes in Ψ can take, where $\nu(\psi)$ is the finite set of values that ψ can take. ■

Intuitively, these graphs encode two forms of variability. The first consists of variations in the data flow, as we allow data paths to have multiple connected producers and consumers; this follows the same approach as in variable workflows [57]. The second lies in the alternative attribute values that a node can take. For example, consider again Fig. 2. We see that datum `D2` can have three `size` values. This corresponds to a design variation of the application (i.e. the size of the processed data). On the contrary, the `quality` of `D2` represents the impact of data size on the overall quality of the system, which is also determined by the quality value of `B`. Thus, the overall quality depends on *(i)* the size of `D2` and *(ii)* whether `A` or `B` consumes `D1`. In our case study, the property value of the system is obtained by summing the property values of its constituents. We make this assumption in the rest of the paper, without loss of generality: one can use instead other aggregation functions (e.g. average, maximum).

Our definition of χ_d allows one to define cross-cutting constraints over the attribute values of a given node n. For instance, the `quality` value of `D2` is directly linked to its `size` property: a `size` of 256 leads to a `quality` of 0, 512 to 1, and 1024 to 3. We see that $\chi_d(n)$ defines which valuations of the attributes are valid *altogether*. This flexible definition, akin to the notion of configuration of non-boolean parameters [61]–[65], can express that some attribute values are forbidden in n, and that the value of given attributes in n restricts the values of the others.

Definition 2: A variable resource graph is a tuple $VRG = (R, C, \Theta, \Psi, \chi_r)$ where $R = P \cup S$ is a set of resources (P are processors and S are memory storage); $C \subseteq (S \times P) \cup (P \times S)$ is a set of connections between processors and storage, where $(p, s) \in C$ (resp. $s, p \in C$) means that processor p writes from (resp. reads to) storage s; $\Theta : 2^R \rightarrow \{true, false\}$ encodes which subsets of resources constitute a valid platform configuration; Ψ is a set of attributes; $\chi_r : R \rightarrow (\Psi \rightarrow \bigcup_{\psi \in \Psi} \nu(\psi)) \rightarrow \{true, false\}$ associates a resource r to the set of values that the attributes in Ψ can take. ■

The function χ_r is defined similarly to its counterpart in variable data-flow graphs and offers the same benefits.

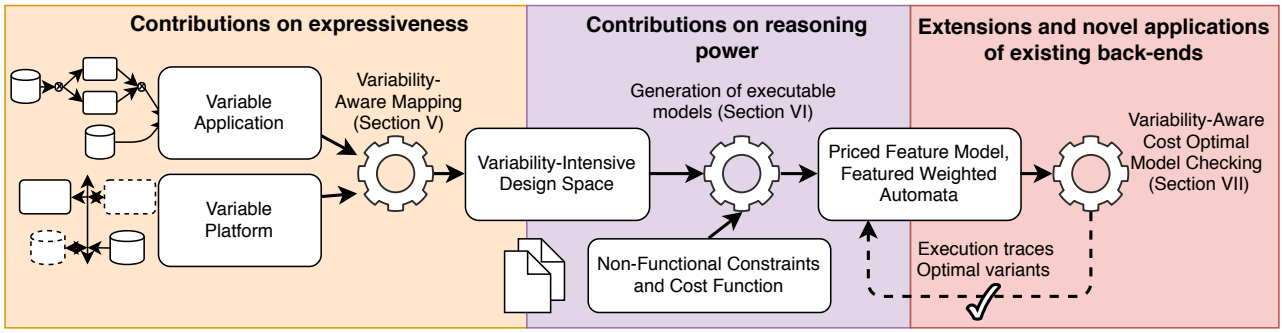

Fig. 4: Framework Models and Processes

Another source of variability encoded by this model comes from the optional nature of resources, that is, two variants of the platform may differ by their constituent resources. Θ symbolically encodes which subsets of resources constitute a valid platform in the form of Boolean constraints.

The third and last part of our formal model is the mapping rules that must be satisfied to deploy any given application variant on any given platform variant. These rules are automatically generated from the variable data-flow graph and the variable resource graph. A mapping allocates each task to at least one processor function, each datum to at least one memory storage, and each path to at least one buffer or memory storage. A valid mapping must ensure that the required data can be read and written by the hardware functions using them. For each path p, producer node i and consumer task o of p, there must exist a buffer or storage b associated to p, a function f_i associated to i, and a function f_o associated to o, such that f_i can write to b and f_o can read from b.

Altogether, the two graphs and their mapping rules constitute the *variability-intensive design space*, as they define the set of the design alternatives that can result from a valid deployment of a variant of the application onto a configuration of the platform.

VI. Enabling Reasoning on the Design Space

From the variability-intensive design space, we generate intermediate representations to reason on the structure and the behaviour of all variants and on their quality attributes.

In product-line engineering [66], structural variability is commonly addressed by modelling *features* and their interdependencies as a *feature model* (FM) [67]. Since we consider NF requirements, we use an extension of FMs where features have quality attributes [10], [14], [68], which we name *Priced Feature Model* (PFM). Reasoning on the PFM allows for determining which design variants satisfy the requirements that depend on the structure of the design (FCS and NFS).

Definition 3: A PFM is a tuple $pfm = (F, Q, \Psi_\rho, \eta, \Psi_\tau)$ where F is a set of Boolean features; Q is a set of positive real-valued quality attributes; Ψ_ρ is a set of cross-cutting constraints over F defining which subsets of features form a valid variant; $\eta : F \times Q \to \mathbb{R}_0^+$ is a function defining how each $f \in F$ changes the value of each $q \in Q$; Ψ_τ is a set of constraints over Q defining what are the valid aggregated values for a given attribute q. The semantics of a PFM is a partial function $[\![pfm]\!] : 2^F \to Q \to \mathbb{R}_0^+$ such that *(i)* $[\![pfm]\!]$ is defined only for the valid variants: $F' \in dom([\![pfm]\!]) \Leftrightarrow F' \models \Psi_\rho$; *(ii)* the aggregated value of each attribute q satisfies the constraints: $\forall q \in Q : \eta(F', q) \models \Psi_\tau$; *(iii)* each attribute is associated with its aggregated value: $[\![pfm]\!](F', q) = \eta(F', q)$ where $\eta(F', q)$ is the aggregated value of q in the variant represented by $F' \subseteq F$. ■

An excerpt of PFM is shown in Fig.5. Each variation point that may occur in the application (e.g., consumer of `P1` is `A` or `B`), platform (e.g., `RAM` is present or not) or mapping (e.g., `D1` can be stored in `ROM` or `RAM`) gives rise to an optional feature in the PFM. The attributes representing design variations (e.g. the size of `D2`) are transformed into alternative features, while those representing quality attributes (e.g. cost of manufacturing) become the PFM's quality attributes. Ψ_ρ results directly from the mapping rules (3a and 3b), the presence conditions Θ between resources (5), consistency rules requiring that alternative application node or optional resources used in the mapping become mandatory (1,2,4), and constraints over attribute values encoded in χ_d and χ_r. Ψ_τ correspond to the NF requirements defined by the engineers. Finally, η is obtained from χ_d and χ_r.

To check requirements that depend on behaviour, we need to systematically evaluate all executions of all design variants. A common approach to achieve this for a single design is to model-check an automaton modelling the system behaviour [69], [70], which results from the scheduling [71] of an application automaton on a platform automaton. These approaches, however, cannot handle variability and are thus limited to one system at a time. Accordingly, we propose to generate, from our variability-intensive design space, a *Featured Weighted Automaton* (FWA) [60] – a recent formalism that combines featured transition systems [22] with priced automata [72]. Basically, FWA is a variability-aware, weighted extension of finite state machines where each transition t is annotated with a formula defining (i) which variants can execute t; and (ii) conditional weight values that depend on the variant executing t. This formula, named *weighted feature*

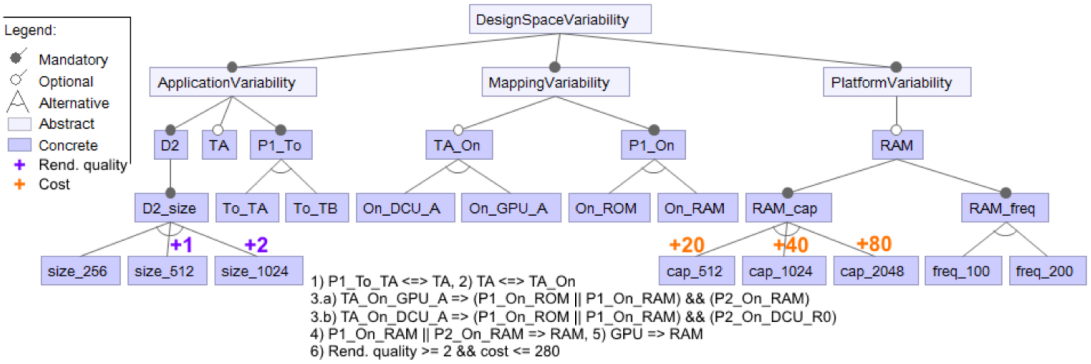

Fig. 5: An excerpt of the PFM corresponding to Fig. 2 and 3.

expression, allows one to represent compactly the evolution of quality attributes as the execution of the variants progresses.

Definition 4: Let F be a set of features and $\{\tau_1 \dots \tau_n\}$ an ordered set of n positive real-valued weights. Then a weighted feature expression is a (possibly partial) function $\gamma : 2^F \to \mathbb{R}^n_{\geq 0}$ that associates a variant $F' \subseteq F$ with a vector w such that w_i is the value of the i-th weight for F'. ■

Intuitively, the weighted feature expression γ associated to a given transition t is such that a variant F' belongs to $dom(\gamma)$ if and only if F' can execute t, and $\gamma(t)$ returns the value of t's weights for F'. In our case, each weight corresponds to a distinct quality attribute and represents the value added to this attribute when F' executes the transition t.

Definition 5: Let $T = \{\tau_1 \dots \tau_n\}$ be a set of positive real-valued, quality attributes and $pfm = (F, Q, \Psi_\rho, \eta, \Psi_\tau)$ be a PFM, such that $Q \subseteq T$. A FWA over T and pfm is a tuple $fwa = (S, s_0, s_f, T, \gamma_\to)$ where S is a set of states, s_0 is the initial state, s_f is the final state, $T \subseteq S \times S$ is the transition relation, and $\gamma_\to : T \to 2^F \to \mathbb{R}^n_{\geq 0}$ associates each transition with a weighted feature expression. ■

A FWA defines, for each variant $F' \subseteq F$, the set of paths starting from s_0 to s_f that F' can execute, together with the weight vector associated to each path: $\forall F' \subseteq F : [\![fwa]\!](F') = \{(p, w) \in S^+ \times \mathbb{R}^n_{\geq 0}\}$ where $p = s_0 \to \dots \to s_k = s_f$ and such that *(i)* p exists in fwa: $\forall j : 0 \leq j < k : (s_j, s_{j+1}) \in T$; *(ii)* all transitions of p are executable by F': $F' \in \bigcap_{j=0}^{k-1} dom(\gamma_\to(s_j, s_{j+1}))$; *(iii)* w is the sum over all associated weight vectors: $w = \Sigma_{j=0}^{k-1} \gamma_\to(s_j, s_{j+1})$.

Note that we assume all quality attribute values to be positive. Negative values are supported modulo transformation. In PFM, one can replace negative values on a feature by positive values on the alternative features. In FWA, one can replace negative values on a transition by positive values on the alternative transitions.

From the design space, we generate a network of FWA, where each source datum, task, processor and memory storage corresponds to a distinct FWA whose states and transitions encode the different steps of their process. Data automata implement the process of sending the data onto a task automaton, which itself transmits the data to the automaton of the processor that provides the function whereon the task is mapped. The automaton of a processor models standard processing pipeline behaviour [23]. During this processing, the processor can store input and output data by transferring them to its dedicated storage automaton.

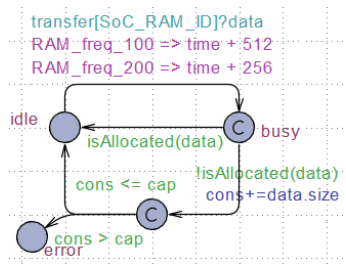

Fig. 6: An FWA modelling memory storage behaviour.

An excerpt of FWA modelling storage behaviour is shown in Fig. 6. It models memory read and write via input/output interfaces while checking that memory capacity is never exceeded. Weighted feature expressions occur in the transitions of the hardware resources automata (i.e. processors and memories) that correspond to processing instructions and memory accesses, as these operations impact the "execution time" quality attribute. In the excerpt, such an expression is shown in pink and models that the increase of time depends on RAM frequency. The automaton modelling all the designs results from the parallel composition of the individual automata, which can synchronize their transitions on specific synchronization actions (e.g. the `transfer` action in Fig. 6 sends `data` only when another FWA executes the corresponding action).

Our FWA differs from that of Fahrenberg et al. [60]: *(i)* ours is linked to a PFM to also check structural requirements; *(ii)* we support parallel composition and consider time as a special weight in that we execute fastest actions first and dynamically update the remaining time delay of other parallel actions to simulate time elapsing; *(iii)* Fahrenberg et al. [60] represents $\gamma_\to$ as a partition of the set of variants because their verification algorithm – which is not tooled – relies on matrix representations, whereas ours relies on antichains [73] to scale wrt. the large state space incurred by our evaluation cases.

VII. All-In-One Verification and Optimization

Our FWA formalism allows us to design a first algorithm to solve all challenges FCS, FCB, NFS and NFS for all variants at once. The key idea is to perform an exploration of the state space of the automaton in search of paths that can reach the final state while satisfying the FC and NF requirements.

The first step filters out variants that do not satisfy PFM constraints Ψ_ρ and Ψ_τ, thereby ensuring that the structure of the variants does not violate the requirements. Then we explore all paths starting from the initial state. As we visit a new state, we retain the set of variants able to execute the sequence of transitions that led to the state. We also accumulate the sum of the weights over all executed transitions and for all variants and assert that these values satisfy the NF requirements. In the end, we obtain a set of paths going from the initial to the final state, together with, for each path p, (a) the valid variants that can execute p and (b) the values of the quality attributes corresponding to p and each of those variants.

This first algorithm finds all variants satisfying the requirements. We have to find what are the *optimal* variants satisfying the requirements while providing the best values for the quality attributes. Since these attributes can be antagonistic (e.g., manufacturing costs can decrease to the detriment of rendering quality), this is assimilated to multi-objective optimization. To drive our search for optima, we define a cost function over the attributes: let $\zeta(\tau_1, \ldots, \tau_n) = \theta_1 \times \tau_1 \ldots \theta_n \times \tau_n$ be our cost function, such that $\theta_i \in \mathbb{R}^+$ is the coefficient associated to the attribute τ_i. Then, our objective is to discover the variant that minimizes ζ. This is achieved by modifying our exploration algorithm in order to *(i)* record the optimal variants and their associated attribute values, and *(ii)* stop exploring a path as soon as all quality attributes reach a worse (i.e. higher) value. This latter heuristics requires the cost function to be monotonic as more states are explored along a given path; hence why we assume that all θ_i and τ_i are ≥ 0. Negative values can be supported by disabling the heuristics (Line 7 in Algorithm 1).

Algorithm 1 details this exploration procedure. It takes as input a FWA, a PFM and a cost function ζ. It iteratively computes R, the reachability relation that associates each state s to the γ function encoding all variants that can reach s and their associated attribute values. At first (Line 1), R contains s_0 together with γ_0, such that $dom(\gamma_0) = dom([\![pfm]\!])$ and $\gamma_0(F', \tau_i) = 0$ for any variant F' and attribute τ_i. Then, we start the exploration from s_0 (L3–4) and iterate over the states encountered successively (L5–20). At each iteration, we retrieve the state s reached last, together with its associated γ function (L6). Here, γ encodes the variants that can reach s and associates to each variant the values of its attribute values when following the path that led to s. If all these variants yield a value for ζ greater than the current optimum ζ^* (L7), we do not pursue the exploration further from this state. Otherwise, we distinguish between the cases where s is s_f (L8–10) and where it is not (L11–18). In the first case, we assign ζ^* to the minimal cost over all variants that can reach the final state. In the second case, we compute the set of successors of (s, γ)

Input: $fwa = (S, s_0, s_f, \rightarrow, \gamma_\rightarrow)$;
$pfm = (F, Q, \Psi_\rho, \eta, \Psi_\tau)$; $\zeta : \zeta(\tau_1, \ldots, \tau_n) \in \mathbb{R}^+$
Output: $\mathcal{F}^*$, the set of optimal variants that reach s_f
ζ^*, their associated minimal cost

1 $R \leftarrow \{(s_0, \gamma_0)\}$;
2 $\zeta^* \leftarrow +\infty$;
3 $Stack \leftarrow []$;
4 $push(\{s_0, \gamma_0\}, Stack)$;
5 **while** $Stack \neq []$ **do**
6 $\quad (s, \gamma) \leftarrow pop(Stack)$;
7 $\quad$ **if** $\exists F' \in dom(\gamma) : \zeta(\gamma(F')) \leq \zeta^*$ **then**
8 $\quad\quad$ **if** $s = s_f$ **then**
9 $\quad\quad\quad \zeta^* \leftarrow \min_{F' \in dom(\gamma)} \zeta(\gamma(F'))$;
10 $\quad\quad$ **end**
11 $\quad\quad$ **else**
12 $\quad\quad\quad$ **foreach** $(s', \gamma') \in Post(s, \gamma)$ **do**
13 $\quad\quad\quad\quad$ **if** $\nexists (s', \gamma'') \in R : \gamma' \succeq \gamma''$ **then**
14 $\quad\quad\quad\quad\quad push((s', \gamma'), Stack)$;
15 $\quad\quad\quad\quad\quad R \leftarrow R \sqcup \{(s', \gamma')\}$;
16 $\quad\quad\quad\quad$ **end**
17 $\quad\quad\quad$ **end**
18 $\quad\quad$ **end**
19 $\quad$ **end**
20 **end**
21 $\mathcal{F}^* \leftarrow \{F^* \subseteq F | (s_f, \gamma_f) \in R \wedge \gamma_f(F^*) = \zeta^*\}$;
22 **return** $(\mathcal{F}*, \zeta^*)$

Algorithm 1: optima(fwa, pfm, ζ)

(L12), given by $Post(s, \gamma) = \{(s', \gamma') | (s, s') \in T \wedge \gamma' = \gamma \otimes \gamma_\rightarrow(s, s')\}$ where $dom(\gamma_1 \otimes \gamma_2) = dom(\gamma_1) \cap dom(\gamma_2)$ and $\forall F' \in dom(\gamma_1 \otimes \gamma_2) : (\gamma_1 \otimes \gamma_2)(F') = \gamma_1(F') + \gamma_2(F')$. This means that γ' is defined only for the variants that can reach s and execute the transition from s to s', and it sums the attribute values of each variant in γ with its attribute values on the transition (s, s'). Then, we add a successor iff it *improves* the reachability relation (L13), that is, if for at least one variant, there is no element in R that gives a better value for all attributes. This rule ensures that infinite cycles are avoided. To do so, we use the comparison operator $\succeq$ over weighted feature expressions, defined as $\gamma_1 \succeq \gamma_2 \equiv \forall F' \in dom(\gamma_1) : \gamma_1(F') \geq \gamma_2(F')$. If R is improved, we continue the exploration (L14) and add the successor to R (L15) using a particular union operator $\sqcup$ that keeps R as an antichain. This is achieved by a split-and-combine algorithm along the lines of [24], [60], which we do not detail due to lack of space. Finally, we return the set of variants $\mathcal{F}^*$ that can reach s_f while minimizing ζ, together with the optimal cost (L21–22).

VIII. Implementation and Evaluation

A. Implementation

We implemented our framework as a toolchain combining a new Java tool with (extensions of) existing model checkers.

Our Java tool[1] consists of two modules, each of which implements a process depicted in Fig. 4. The first one allows for specifying a variable data-flow graph and a variable resource graph via a fluent API. Then, it calls our mapping algorithm to generate the design space. Second, an automata generator takes as inputs the design space, NF requirements and the cost function to generate an FWA with its associated PFM. Two concrete syntaxes are used, as we can then invoke two independent model checkers to search for optima.

One is UPPAAL-CORA [32], an established tool to carry out cost-optimal reachability analyses that we reuse as is with optimal settings. It takes as input a network of Linearly Priced Timed Automata (LPTA) [74]. LPTA can be regarded as FWA without variability, and as such can only encode the behaviour of the variants separately. Our automata generator actually transforms our design space into a network of LPTA in the UPPAAL-CORA format. It also generates an additional automaton dedicated to configuring the other LPTAs *before* their execution starts, by setting variables that correspond to the variation points of the design space. We thus follow the 150% model approach [75]. Additionally we use SPLOT's feature model reasoning library [76] to restrict the configuration to valid products. Then, UPPAAL-CORA can find an execution of a variant that reaches the accepting state while satisfying all the NF requirements and optimizing the cost function.

The other model checker is ProVeLines [77], which can check variability-intensive systems. We chose this tool because it was extended over the years, by both its original developers [77] and others [19], [29], to solve multiple model-checking problems including real-time verification [24]. We fully implemented Alg. 1 in a new version of ProVeLines.[2] To achieve this, we first extended ProVeLines' input language – Promela [78] – to associate Promela statements with weighted feature expressions. Actually, each Promela process encodes a single FWA. Like UPPAAL-CORA, our ProVeLines extension is able to provide the execution trace associated to an optimal variant. The difference lies in that weighted feature expressions allow an all-at-once verification of all variants. To encode the structural variability, we generate a PFM in the format supported by ProVeLines, viz. TVL [64], [79].

B. Qualitative Evaluation

Our first evaluation assesses the usefulness of our approach for practitioners on the basis of Visteon's instrument cluster system (see Sec. II). More precisely, our evaluation concerns an important module of the whole system. Yet, it remains a real-world case that was selected by our partner as representative in terms of size, complexity and variability.

With the assistance of an expert engineer, we reverse-modelled the variable application and platform of the existing system. Some technical simplifications were made as we aim at facilitating early design decisions: we do not consider data and parameters that only have minor impact on the system's runtime; we abstract away from data content and consider data sizes as the most influential factor for runtime performance; we do not model mechanisms like internal cache replacement policies, AXI bus and internal backup communication buffers as these implementation details have no fundamental impacts and are handled by engineers later in the development process. These simplifications were deemed harmless by our industrial expert and did not endanger the correctness of our results.

This results in an application model with two input data types, four tasks and 32 flow processing, and in a platform model with one ROM, one RAM, one DCU and two GPUs. In total, the variability yields 1,548,288 candidate designs. Our generated mapping rules reduced this number to 1,878. By adding the NF requirements *rendering quality* $\geq$ *2* and *manufacturing cost* $\leq$ *280*, we further diminish this number to 894. By checking the behaviour against the requirements (i.e. the end of execution is reached within 840 processing cycles) we obtain 279 variants that satisfy all requirements. Incorporating the cost function representing trade-offs defined with the expert (i.e. with θ_{time} = 1, $\theta_{m.cost}$ = 10, $\theta_{r.qual.} = 100$) yielded 6 optimal variants with time = 642, manufacturing cost = 140 and rendering quality = 2.

We performed the behavioural analyses using both UPPAAL-CORA and ProVeLines, which provided the same results for all variants. This increases our confidence that the transformation of the design space into LPTA and Promela is consistent. Expert's confirmation is also needed, though, as mistakes may originate from the input models themselves. The expert validated that the optima returned by our tools conform to the very best designs that the company could produce over the past years. The quality attributes' values also corresponded to what is expected. Regarding execution time (cycles), there are slight differences due to hardware modelling simplifications; however, the numbers are close to reality ($<$ 10% difference) and the relative orders between variants are preserved. In the end, the expert validated that our approach helps make optimal design decisions that will provide significant gains at all stages of development.

C. Quantitative Evaluations

The second part of our evaluation focuses on the efficiency and the scalability of our approach in terms of execution time. This is indeed essential, as the total number of variants can be high in real-world systems. In addition to the instrument cluster case study (numbered case #0), we constituted a dataset of realistic topologies that were generated based on our partner's history. To do so, our model generator relies on multivariate Gaussian distributions whose parameters were settled on the basis of Visteon's past systems. Thereby, it ensures that the characteristics of our generated data-flow and platform models are similar to real-world cases. Amongst all the models we generated, we selected 11 of them that appropriately summarize our findings. These models exhibit different state densities (i.e. average number of system states per variant) and variability intensities (i.e. numbers of valid variants to check). We carried out three series of experiments

[1] https://bitbucket.org/SamiLazreg/enlighter

[2] https://bitbucket.org/maxcordy/provelines-cora

TABLE I: Results for the three quantitative evaluations. Times are in seconds.

			(1) FCS + FCB + NFS + NFB				(2) NFOB		(3) NFOS Priorization		
			Product-based		Family-based		P.V.L. optim	UPP.-CORA	ClaferMOO + P.V.L.		
case	density	variants	time	explored	time	explored	time	time	time	# OOS	inc. %
Ins. Cl. (#0)	369	1,878	22.60	693,178	3.33	305,114	2.42	5.64	0.08	10	74.10%
Gen. #1	1,561	32	0.80	49,952	0.48	45,681	0.38	18.61	=	=	=
Gen. #2	2,250	64	2.72	143.968	2.10	124,004	1.27	39.59	=	=	=
Gen. #3	3,061	243	22.11	743,823	80.54	660,348	75.79	OoM.	=	=	=
Gen. #4	1,656	516	16.32	854,996	10.95	777,616	8.23	17.60	2.32	129	9.19%
Gen. #5	2,256	1,152	55.48	2,599,393	40.17	2,217,593	29.77	OoM.	=	=	=
Gen. #6	1,496	1,280	38.49	1,915,264	11.81	840,711	8.65	19.89	1.37	64	40.86%
Gen. #7	2,416	2,187	144.48	5,283,792	120.83	4,593,249	89.56	OoM.	=	=	=
Gen. #8	1,461	2,592	109.96	3,785,940	35.01	1,032,639	24.40	OoM.	17.60	324	1.39%
Gen. #9	2,273	3,168	151.87	7,201,320	69.14	3,926,395	19.58	OoM.	6.21	792	9.81%
Gen. #10	1,609	12,288	395.64	19,777,536	148.71	6,570,642	75.97	OoM.	3.62	256	64,90%
Gen. #11	1,732	34,560	1344.23	59,858,304	227.31	10,871,741	72.17	OoM.	2.90	32	95.77%

presented hereunder. Table I provides the results, such that the results of the different series of experiments are separated by double borders. All benchmarks were run on a MacBook Pro 2014 with a 2,8 GHz Intel Core i7 processor and 16 GB of DDR3 RAM. We repeated each experiment ten times and computed the average, although random deviations were low.

Product-based vs. family-based. Our first experiments evaluate the efficiency of our method to verify that *all* variants satisfy the requirements, that is, we consider only the four challenges FCS, FCB, NFS and NFB. We compare the runtime of our family-based verification algorithm with an alternative, product-based one that checks each variant separately. Both algorithms are implemented in ProVeLines, which allows us to compare both approaches on an equal technological ground.

The results are presented on the left part of Table I. It depicts, for each approach and model, the time needed to check all variants as well as the total number of states that were explored by each algorithm. In the family-based case, fewer states are explored since one state common to multiple variants is explored only once. For case #0, the family-based method outperforms the product-based one, reducing the verification time from 22.60 seconds to 3.33. The generated models allow us to observe that the benefits of the family-based method grow with the number of variants. When this number is low (#1–3), our family-based algorithm either brings insignificant improvements (#1 and #2) or performs way worse than the product-based approach (#3). On the contrary, for models with higher variability (#4–11) we obtain reductions in verification time of minimum 16% (#6), most often substantial ones. The most impressive results are obtained for the case with the most variants (#11 – which is also the case where variants share the most commonalities): our algorithm is 5.9 times faster, achieving an absolute reduction of 1,116.92 seconds. We also see that a higher state density often reduces the gain offered by our algorithm (e.g. cases #5 and #7). To explain this, we analyzed the models and observed that a small number of variants exhibit a large state space, while all the others encompass far fewer states. The variants thus have fewer states in common, while family-based algorithms generally perform better as variants share more commonalities. This also explains the poor performance of our algorithm in case #3.

Optimization in ProVeLines and UPPAAL-CORA. Our second experiment compares the efficiency of the two model checkers to solve challenge NFOB, that is, we compare UPPAAL-CORA cost-optimal reachability algorithm against our Algorithm 1. The two tools present clear differences, notably in history (UPPAAL-CORA's development started in the early 2000's, while ProVeLines was released in 2013), in focus (cost-optimal reachability in continuous-time models vs. family-based model checking in discrete-time models) and in input syntax (LPTA vs. Promela). Still, we believe this comparison can highlight interesting research directions.

Results are given at the centre of Table I. Like any product-by-product approach, UPPAAL-CORA suffers from every increase in the number of variants, except for cases #4 and #6 where it luckily alleviates complexity with branch-and-bound optimizations. It systematically performs poorer than ProVeLines. Even worse, apart from cases #0–2, #4 and #5, UPPAAL-CORA consumes too much memory ($>$ 16GB) and raises a fatal error. We assumed two reasons behind this. First, UPPAAL-CORA does not implement partial-order reduction [80] and thus considers all possible interleavings between LPTAs, including during their configuration. This creates an exponential blow-up as the number of LPTAs increases. Second, we use a model that encodes the behaviour of all variants and thus accumulates all their state space. Yet, applying an alternative, product-based approach where each variant is turned into a separate model did not solve the problem and even led to slower times. Process interleaving is therefore responsible. One way to circumvent this is to assign priorities over the automata. However, in most cases, this will cause to miss better scheduling opportunities and will yield suboptimal results. In spite of the disappointing results for UPPAAL-CORA, the fact that it outperforms product-based ProVeLines on case #0 tends to indicate that combining our family-based algorithm with UPPAAL-CORA's efficient search heuristics can be a promising future work.

Through these experiments, we also observe that looking only for optima in ProVeLines, as opposed to verifying all variants, can yield significant reductions in execution time (up to 68% in case #11). More importantly, computing the optima without a family-based algorithm boils down to applying the

product-based algorithm used in the first series of experiments. In this regard, our Algorithm 1 exacerbates the benefits of a family-based algorithm. For the models with the most variants (#9–11), Algorithm 1 is 5 to 128 times faster than the product-based method. Interestingly, it seems to be way less affected by the number of variants than the all-variant verifications.

Prioritization based on structural optima. The last part of our evaluation studies whether the structural optima (i.e. the solutions of NFOS) are sufficiently close to the overall optima (i.e. the solutions of NFOB). If that is the case, solving NFOS would yield an imperfect but relevant, cost-effective solution, as structural optimality can be resolved statically, that is, without requiring the exploration of a large state space. We use the ClaferMOO tool [14], [81] to compute the structural optima based on our PFMs. To complete the toolchain, we extended it to generate PFMs in ClaferMOO's format, as well as our verifier module to retrieve the variants that are structural optima and only allow these variants by adding a constraint in the PFM. Then, we use ProVeLines in family-based mode to assess the time needed to discover which structural optima lead to the lowest cost function value, as well as the difference between this value and that of the overall optimum.

Results are given on the right side of Table I. The first two columns give the number of structural optima returned by ClaferMOO, as well as the time needed to run ProVeLines only on those structural optima. The last column gives the increase of cost function, in percentage, due to considering only structural optima instead of all variants. In cases #1–3, #5 and #7, the variants differ only by their behaviour and are thus all structural optima. For the other cases, we see that ProVeLines' runtime is dramatically decreased as it has to check fewer variants. This, however, comes at the cost of a notable increase in cost function value (up to 96%), except for case #8 where it increases by less than 2%. We conclude that structural optima are still far from being overall optima, and thus are not sufficient. Yet, in some cases they can constitute viable solutions to get a quick answer or when the state space of the system is too large to be exhaustively explored.

D. Threats to Validity

Internal. The main threat to internal validity is the selection of the used case study. Although it was built from specification and feedback meetings with system and platform experts from our partner company, it is a single case. Still, Visteon chose it as being representative in terms of shape and complexity of the data flow, the platform and their variability.

External. Several threats to external validity exist. First, our scalability experiments have been conducted over large but simulated data sets. We nevertheless expect the creation procedures to respect the structure and behaviour of the potential applications and platforms of our industrial partner. More generally, we also expect the proposed approach to be applicable in other domains where data flows can be an appropriate modelling support, like stateless enterprise processes with micro-services. On the platform side, the component-based representation seems general enough to cover different forms of deployment architectures, but we do not have any evidence yet of this generalization possibility.

Second, satisfiability and optimality checking processes depend highly on the level of detail and on the quality of application and platform modelling. On the platform side, a design not detailed enough could lead to loss of relevance in the performance metrics, and could even produce false optimal results. From the feedback we got from our industrial partner, the level of detail is currently sufficient as it is at the level they use to specify the platform when selecting suppliers. Still, in other contexts, some details may be hidden due to intellectual property management. As for applications, they are described with data-flows that are well mastered by *domain experts* in our case study, but they have been built manually. Reverse engineering from existing application code could be envisaged to build data-flow or at least templates of them, but their quality would be directly related to the good organization of existing software. More globally, as the engineers were able to understand proposed designs that result from our framework, machine learning technique fed by their feedback on good or bad solutions could help when details are missing. Introducing this in the framework remains an open issue.

IX. Conclusion

In many data-flow oriented embedded systems, three levels of variability significantly increase the complexity of the design space: variable high-level data-flows are deployed in many different ways over highly configurable component-based hardware platforms. We provided a modelling and reasoning framework that unifies state-of-the-art techniques on structural reasoning with a novel variability-aware model-checking algorithm, which evaluates the functional feasibility, the non-functional satisfiability and optimality on the whole design space. This design space comes as a mapping between two other models: one represents the variable applications with a data-flow model complemented with quality attributes, the other models platform variability through connected components, also with quality attributes. We showed how the design space is transformed into featured priced timed automata to enable different reasoning and optimization operations.

The application of the proposed approach to a medium-scale industrial case of an automotive instrument cluster demonstrates its end-to-end ability to check and optimize a complex design space. System experts revealed that, even on small data flow and platform models, the optimal designs were non-trivial to identify for them, giving us confidence on the relevance of the approach. Experiments on large simulated design spaces also show that the prototyped toolchain exhibits good scalability and outperforms non-variability-aware solutions.

We believe that the models used in the framework (data-flows and components) make the contribution potentially applicable in different contexts. Our future works will start with an extension of the framework to facilitate its usage, introducing domain-specific languages for input models. We also plan to conduct larger evaluations on different data flows coming from various product lines of our industry partners.

REFERENCES

[1] A. Pretschner, M. Broy, I. H. Kruger, and T. Stauner, "Software engineering for automotive systems: A roadmap," in *2007 Future of Software Engineering*, ser. FOSE '07. IEEE Computer Society, 2007, pp. 55–71.

[2] M. Broy, S. Chakraborty, D. Goswami, S. Ramesh, M. Satpathy, S. Resmerita, and W. Pree, "Cross-layer analysis, testing and verification of automotive control software," in *Proceedings of the 11th International Conference on Embedded Software, EMSOFT 2011, part of the Seventh Embedded Systems Week, ESWeek 2011, Taipei, Taiwan, October 9-14, 2011*, 2011, pp. 263–272.

[3] M. Broy, "Challenges in automotive software engineering," in *28th International Conference on Software Engineering (ICSE 2006), Shanghai, China, May 20-28, 2006*, 2006, pp. 33–42.

[4] F. Balarin, *Hardware-software co-design of embedded systems: the POLIS approach*. Springer Science & Business Media, 1997.

[5] B. Kienhuis, E. Deprettere, K. Vissers, and P. Van Der Wolf, "An approach for quantitative analysis of application-specific dataflow architectures," in *Application-Specific Systems, Architectures and Processors, 1997. Proceedings., IEEE International Conference on*. IEEE, 1997, pp. 338–349.

[6] E. Khalilov, J. Ross, M. Antkiewicz, M. Völter, and K. Czarnecki, "Modeling and optimizing automotive electric/electronic (e/e) architectures: Towards making clafer accessible to practitioners," in *International Symposium on Leveraging Applications of Formal Methods*. Springer, 2016, pp. 447–464.

[7] J. A. Ross, A. Murashkin, J. H. Liang, M. Antkiewicz, and K. Czarnecki, "Synthesis and exploration of multi-level, multi-perspective architectures of automotive embedded systems," *Software & Systems Modeling*, pp. 1–29, 2017.

[8] J. Guo, D. Yang, N. Siegmund, S. Apel, A. Sarkar, P. Valov, K. Czarnecki, A. Wasowski, and H. Yu, "Data-efficient performance learning for configurable systems," *Empirical Software Engineering*, pp. 1–42, 2017.

[9] Y. Zhang, J. Guo, E. Blais, and K. Czarnecki, "Performance prediction of configurable software systems by fourier learning (t)," in *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on*. IEEE, 2015, pp. 365–373.

[10] N. Siegmund, M. Rosenmüller, M. Kuhlemann, C. Kästner, S. Apel, and G. Saake, "Spl conqueror: Toward optimization of non-functional properties in software product lines," *Software Quality Journal*, vol. 20, no. 3-4, pp. 487–517, 2012.

[11] N. Siegmund, A. Grebhahn, S. Apel, and C. Kästner, "Performance-influence models for highly configurable systems," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. ACM, 2015, pp. 284–294.

[12] P. Jamshidi, N. Siegmund, M. Velez, C. Kästner, A. Patel, and Y. Agarwal, "Transfer learning for performance modeling of configurable systems: An exploratory analysis," in *Proceedings of the 32Nd IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE 2017. Piscataway, NJ, USA: IEEE Press, 2017, pp. 497–508. [Online]. Available: http://dl.acm.org/citation.cfm?id=3155562.3155625

[13] P. Valov, J.-C. Petkovich, J. Guo, S. Fischmeister, and K. Czarnecki, "Transferring performance prediction models across different hardware platforms," in *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering*. ACM, 2017, pp. 39–50.

[14] R. Olaechea, S. Stewart, K. Czarnecki, and D. Rayside, "Modelling and multi-objective optimization of quality attributes in variability-rich software," in *Proceedings of the Fourth International Workshop on Non-functional System Properties in Domain Specific Modeling Languages*, ser. NFPinDSML '12. ACM, 2012, pp. 2:1–2:6.

[15] E. Zulkoski, C. Kleynhans, M.-H. Yee, D. Rayside, and K. Czarnecki, "Optimizing alloy for multi-objective software product line configuration," in *Proceedings of the 4th International Conference on Abstract State Machines, Alloy, B, TLA, VDM, and Z - Volume 8477*, ser. ABZ 2014. Springer-Verlag New York, Inc., 2014, pp. 328–333.

[16] R. Olaechea, D. Rayside, J. Guo, and K. Czarnecki, "Comparison of exact and approximate multi-objective optimization for software product lines," in *SPLC '14*. ACM, 2014, pp. 92–101.

[17] S. Kugele and G. Pucea, "Model-based optimization of automotive e/e-architectures," in *Proceedings of CSTVA 2014*. ACM, 2014, pp. 18–29.

[18] C. Henard, M. Papadakis, M. Harman, and Y. Le Traon, "Combining multi-objective search and constraint solving for configuring large software product lines," in *Proceedings of ICSE '15*. IEEE Press, 2015, pp. 517–528.

[19] R. Olaechea, U. Fahrenberg, J. M. Atlee, and A. Legay, "Long-term average cost in featured transition systems," in *Proceedings of the 20th International Systems and Software Product Line Conference*, ser. SPLC '16. New York, NY, USA: ACM, 2016, pp. 109–118. [Online]. Available: http://doi.acm.org/10.1145/2934466.2934473

[20] P. Asirelli, M. H. ter Beek, S. Gnesi, and A. Fantechi, "Formal description of variability in product families," in *Software Product Line Conference (SPLC), 2011 15th International*. IEEE, 2011, pp. 130–139.

[21] M. Chechik, B. Devereux, S. M. Easterbrook, and A. Gurfinkel, "Multi-valued symbolic model-checking," *ACM Trans. Softw. Eng. Methodol.*, vol. 12, no. 4, pp. 371–408, 2003.

[22] A. Classen, M. Cordy, P.-Y. Schobbens, P. Heymans, A. Legay, and J.-F. Raskin, "Featured transition systems: Foundations for verifying variability-intensive systems and their application to ltl model checking," *IEEE Transactions on Software Engineering*, vol. 39, no. 8, pp. 1069–1089, 2013.

[23] S. Lazreg, P. Collet, and S. Mosser, "Assessing the functional feasibility of variability-intensive data flow-oriented systems," in *Symposium on Applied Computing*, 2018.

[24] M. Cordy, P.-Y. Schobbens, P. Heymans, and A. Legay, "Behavioural modelling and verification of real-time software product lines," in *Proceedings of the 16th International Software Product Line Conference-Volume 1*. ACM, 2012, pp. 66–75.

[25] G. Cledou, J. Proença, and L. Soares Barbosa, "Composing families of timed automata," in *Fundamentals of Software Engineering*, M. Dastani and M. Sirjani, Eds. Cham: Springer International Publishing, 2017, pp. 51–66.

[26] L. Luthmann, A. Stephan, J. Bürdek, and M. Lochau, "Modeling and testing product lines with unbounded parametric real-time constraints," in *Proceedings of the 21st International Systems and Software Product Line Conference-Volume A*. ACM, 2017, pp. 104–113.

[27] V. Nunes, P. Fernandes, V. Alves, and G. N. Rodrigues, "Variability management of reliability models in software product lines: An expressiveness and scalability analysis," in *SBCARS '12*, 2012, pp. 51–60.

[28] G. N. Rodrigues, V. Alves, V. Nunes, A. Lanna, M. Cordy, P. Schobbens, A. M. Sharifloo, and A. Legay, "Modeling and verification for probabilistic properties in software product lines," in *16th IEEE International Symposium on High Assurance Systems Engineering, HASE 2015, Daytona Beach, FL, USA, January 8-10, 2015*, 2015, pp. 173–180.

[29] R. Olaechea, J. Atlee, A. Legay, and U. Fahrenberg, "Trace checking for dynamic software product lines," in *Proceedings of the 13th International Conference on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS '18. ACM, 2018, pp. 69–75.

[30] G. Behrmann, K. G. Larsen, and J. I. Rasmussen, "Priced timed automata: Algorithms and applications," in *International Symposium on Formal Methods for Components and Objects*. Springer, 2004, pp. 162–182.

[31] P. Bouyer, F. Cassez, E. Fleury, and K. G. Larsen, "Optimal strategies in priced timed game automata," in *International Conference on Foundations of Software Technology and Theoretical Computer Science*. Springer, 2004, pp. 148–160.

[32] G. Behrmann, K. G. Larsen, and J. I. Rasmussen, "Optimal scheduling using priced timed automata," *ACM SIGMETRICS Performance Evaluation Review*, vol. 32, no. 4, pp. 34–40, 2005.

[33] N. Bertrand, S. Pinchinat, and J.-B. Raclet, "Refinement and consistency of timed modal specifications," in *International Conference on Language and Automata Theory and Applications*. Springer, 2009, pp. 152–163.

[34] F. Cassez, P. G. de Aledo, and P. G. Jensen, "Wuppaal: Computation of worst-case execution-time for binary programs with uppaal," in *Models, Algorithms, Logics and Tools*. Springer, 2017, pp. 560–577.

[35] J. Liebig, S. Apel, C. Lengauer, and T. Leich, "Robbydbms: a case study on hardware/software product line engineering," in *Proceedings of the First International Workshop on Feature-Oriented Software Development*. ACM, 2009, pp. 63–68.

[36] T. Fischer, C. Kollner, M. Hardle, and K. D. Muller-Glaser, "Product line development for modular fpga-based embedded systems," in *Rapid System Prototyping (RSP), 2014 25th IEEE International Symposium on*. IEEE, 2014, pp. 9–15.

[37] C. Brink, E. Kamsties, M. Peters, and S. Sachweh, "On hardware variability and the relation to software variability," in *Software Engineering and Advanced Applications (SEAA), 2014 40th EUROMICRO Conference on*. IEEE, 2014, pp. 352–355.

[38] D. Streitferdt, P. Sochos, C. Heller, and I. Philippow, "Configuring embedded system families using feature models," in *Proc. of Net. ObjectDays*, 2005, pp. 339–350.

[39] R. Adler, I. Schaefer, T. Schuele, and E. Vecchié, "From model-based design to formal verification of adaptive embedded systems," in *Proceedings of the formal engineering methods 9th international conference on Formal methods and software engineering*, ser. Proceedings of ICFEM '07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 76–95.

[40] R. Adler, I. Schaefer, M. Trapp, and A. Poetzsch-Heffter, "Component-based modeling and verification of dynamic adaptation in safety-critical embedded systems," *ACM Trans. Embedded Comput. Syst.*, vol. 10, no. 2, pp. 20:1–20:39, 2010.

[41] P. Marwedel, *Embedded system design: Embedded systems foundations of cyber-physical systems*. Springer Science & Business Media, 2010.

[42] T. A. Henzinger and J. Sifakis, "The embedded systems design challenge," in *International Symposium on Formal Methods*. Springer, 2006, pp. 1–15.

[43] J. Eker, J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong, "Taming heterogeneity-the ptolemy approach," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 127–144, 2003.

[44] A. Bakshi, V. K. Prasanna, and A. Ledeczi, "Milan: A model based integrated simulation framework for design of embedded systems," *ACM Sigplan Notices*, vol. 36, no. 8, pp. 82–93, 2001.

[45] A. Davare, D. Densmore, T. Meyerowitz, A. Pinto, A. Sangiovanni-Vincentelli, G. Yang, H. Zeng, and Q. Zhu, "A next-generation design framework for platform-based design," in *Conference on using hardware design and verification languages (DVCon)*, vol. 152, 2007.

[46] T. Basten, E. Van Benthum, M. Geilen, M. Hendriks, F. Houben, G. Igna, F. Reckers, S. De Smet, L. Somers, E. Teeselink *et al.*, "Model-driven design-space exploration for embedded systems: The octopus toolset," *Leveraging Applications of Formal Methods, Verification, and Validation*, pp. 90–105, 2010.

[47] A. D. Pimentel, C. Erbas, and S. Polstra, "A systematic approach to exploring embedded system architectures at multiple abstraction levels," *IEEE Transactions on Computers*, vol. 55, no. 2, pp. 99–112, 2006.

[48] M. Chadli, J. H. Kim, K. G. Larsen, A. Legay, S. Naujokat, B. Steffen, and L.-M. Traonouez, "High-level frameworks for the specification and verification of scheduling problems," *International Journal on Software Tools for Technology Transfer*, pp. 1–26, 2017.

[49] V.-M. Sima and K. Bertels, "Runtime decision of hardware or software execution on a heterogeneous reconfigurable platform," in *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*. IEEE, 2009, pp. 1–6.

[50] K. Sigdel, M. Thompson, A. D. Pimentel, C. Galuzzi, and K. Bertels, "System-level runtime mapping exploration of reconfigurable architectures," in *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*. IEEE, 2009, pp. 1–8.

[51] G. Palermo, C. Silvano, and V. Zaccaria, "Variability-aware robust design space exploration of chip multiprocessor architectures," in *Design Automation Conference, 2009. ASP-DAC 2009. Asia and South Pacific*. IEEE, 2009, pp. 323–328.

[52] L. Schor, I. Bacivarov, D. Rai, H. Yang, S.-H. Kang, and L. Thiele, "Scenario-based design flow for mapping streaming applications onto on-chip many-core systems," in *Proceedings of the 2012 international conference on Compilers, architectures and synthesis for embedded systems*. ACM, 2012, pp. 71–80.

[53] P. Van Stralen and A. Pimentel, "Scenario-based design space exploration of mpsocs," in *Computer Design (ICCD), 2010 IEEE International Conference on*. IEEE, 2010, pp. 305–312.

[54] S. Wildermann, F. Reimann, J. Teich, and Z. Salcic, "Operational mode exploration for reconfigurable systems with multiple applications," in *Field-Programmable Technology (FPT), 2011 International Conference on*. IEEE, 2011, pp. 1–8.

[55] G. Palermo, C. Silvano, and V. Zaccaria, "Robust optimization of soc architectures: A multi-scenario approach," in *Embedded Systems for Real-Time Multimedia, 2008. ESTImedia 2008. IEEE/ACM/IFIP Workshop on*. IEEE, 2008, pp. 7–12.

[56] J. L. Noir, S. Madelénat, G. Gailliard, C. Labreuche, M. Acher, O. Barais, and O. Constant, "A decision-making process for exploring architectural variants in systems engineering," in *Proceedings of the 20th International Systems and Software Product Line Conference*. ACM, 2016, pp. 277–286.

[57] S. Graf, M. Glaß, D. Wintermann, J. Teich, and C. Lauer, "Ivam: Implicit variant modeling and management for automotive embedded systems," in *Hardware/Software Codesign and System Synthesis (CODES+ ISSS), 2013 International Conference on*. IEEE, 2013, pp. 1–10.

[58] S. Graf, S. Reinhart, M. Glaß, J. Teich, and D. Platte, "Robust design of e/e architecture component platforms," in *Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE*. IEEE, 2015, pp. 1–6.

[59] M. Lukasiewycz, F. Sagstetter, and S. Steinhorst, "Efficient design space exploration of embedded platforms," in *Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE*. IEEE, 2015, pp. 1–6.

[60] U. Fahrenberg and A. Legay, "Featured weighted automata," in *5th IEEE/ACM International FME Workshop on Formal Methods in Software Engineering, FormaliSE@ICSE 2017, Buenos Aires, Argentina, May 27, 2017*, 2017, pp. 51–57. [Online]. Available: https://doi.org/10.1109/FormaliSE.2017.2

[61] G. Fleischanderl, G. E. Friedrich, A. Haselböck, H. Schreiner, and M. Stumptner, "Configuring large systems using generative constraint satisfaction," *IEEE Intelligent Systems*, vol. 13, no. 4, pp. 59–68, Jul. 1998.

[62] D. Sabin and R. Weigel, "Product configuration frameworks-a survey," *IEEE Intelligent Systems and their Applications*, vol. 13, no. 4, pp. 42–49, Jul 1998.

[63] A. Hubaux, Y. Xiong, and K. Czarnecki, "A user survey of configuration challenges in linux and ecos," in *VaMoS '12*. ACM, 2012, pp. 149–155.

[64] M. Cordy, P.-Y. Schobbens, P. Heymans, and A. Legay, "Beyond boolean product-line model checking: Dealing with feature attributes and multi-features," in *ICSE'13*. IEEE, 2013, pp. 472–481.

[65] A. Felfernig, L. Hotz, C. Bagley, and J. Tiihonen, *Knowledge-based Configuration: From Research to Business Cases*, 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2014.

[66] P. C. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*, ser. SEI Series in Software Engineering. Addison-Wesley, August 2001.

[67] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson, "Feature-oriented domain analysis (FODA) feasibility study," Tech. Rep. CMU/SEI-90-TR-21, 1990.

[68] D. Benavides, P. T. Martín-Arroyo, and A. R. Cortés, "Automated reasoning on feature models," in *Proceedings of CAiSE'05*, 2005, pp. 491–503.

[69] J. Bengtsson, K. Larsen, F. Larsson, P. Pettersson, and W. Yi, "Uppaal-a tool suite for automatic verification of real-time systems," *Hybrid Systems III*, pp. 232–243, 1996.

[70] E. M. Clarke, O. Grumberg, and D. Peled, *Model checking*. MIT press, 1999.

[71] A. E. Dalsgaard, M. C. Olesen, M. Toft, R. R. Hansen, and K. G. Larsen, "Metamoc: Modular execution time analysis using model checking," in *OASIcs-OpenAccess Series in Informatics*, vol. 15. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2010.

[72] M. Droste, W. Kuich, and H. Vogler, *Handbook of Weighted Automata*. Springer, 2009.

[73] L. Comtet, *Sperner Systems (7.2)*. Springer Berlin Heidelberg, 1974.

[74] G. Behrmann, A. Fehnker, T. Hune, K. G. Larsen, P. Pettersson, J. Romijn, and F. W. Vaandrager, "Minimum-cost reachability for priced timed automata," in *Proceedings of HSCC '01*, 2001, pp. 147–161.

[75] T. Thüm, S. Apel, C. Kästner, I. Schaefer, and G. Saake, "A classification and survey of analysis strategies for software product lines," *ACM Comput. Surv.*, vol. 47, no. 1, pp. 6:1–6:45, 2014.

[76] M. Mendonca, M. Branco, and D. Cowan, "Splot: software product lines online tools," in *Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications*. ACM, 2009, pp. 761–762.

[77] M. Cordy, A. Classen, P. Heymans, P.-Y. Schobbens, and A. Legay, "Provelines: a product line of verifiers for software product lines," in *Proceedings of the 17th International Software Product Line Conference co-located workshops*. ACM, 2013, pp. 141–146.

[78] G. J. Holzmann, *The SPIN Model Checker: Primer and Reference Manual*. Addison-Wesley, 2004.

[79] A. Classen, Q. Boucher, and P. Heymans, "A text-based approach to feature modelling: Syntax and semantics of tvl," *Science of Computer Programming*, vol. 76, no. 12, pp. 1130–1143, 2011.

[80] P. Godefroid, *Partial-Order Methods for the Verification of Concurrent Systems - An Approach to the State-Explosion Problem*, ser. LNCS. Springer, 1996, vol. 1032.

[81] A. Murashkin, M. Antkiewicz, D. Rayside, and K. Czarnecki, "Visualization and exploration of optimal variants in product line engineering," in *Proceedings of the 17th International Software Product Line Conference*. ACM, 2013, pp. 111–115.