A Formal Modeling and Analysis Framework for Software Product Line of Preemptive Real-Time Systems

Jin Hyun Kim, Axel Legay
Louis-Marie Traonouez
INRIA/IRISA, France
{jin-hyun.kim,axel.legay}@inria.fr
louis-marie.traonouez@inria.fr

Mathieu Acher
University of Rennes 1
France
mathieu.acher@irisa.fr

Sungwon Kang
KAIST
Daejeon, Republic of Korea
sungwon.kang@kaist.ac.kr

ABSTRACT

This paper presents a formal analysis framework to analyze a family of platform products w.r.t. real-time properties. First, we propose an extension of the widely-used feature model, called Property Feature Model (PFM), that distinguishes features and properties explicitly Second, we present formal behavioral models of components of a real-time scheduling unit such that all real-time scheduling units implied by a PFM are automatically composed to be analyzed against the properties given by the PFM. We apply our approach to the verification of the schedulability of a family of scheduling units using the *symbolic* and *statistical* model checkers of UPPAAL.

CCS Concepts

•**Software and its engineering → Software verification and validation; Formal software verification;** *Software creation and management;*

Keywords

Software Product Line Engineering, Scheduling Systems, Model Checking, Platform-constrained

1. INTRODUCTION

Software Product Line Engineering (SPLE) allows reusing software assets by managing the commonality and variability of products. Real-time software products (such as real-time operating systems) are a class of systems for which SPLE techniques have not drawn much attention from researchers, despite the need to efficiently reuse and customize real-time artifacts. A real-time system is a time and resource-constrained system, thus it is indispensable to check if a complete system guarantees its composability over timing requirements concerning resource constraints *whenever* it is deployed with varying sets of resources. The same constraints hold for a Software Product Line (SPL) of real-time system, such that all the products generated from the SPL should satisfy various real-time properties.

©2016 ACM. ISBN 978-1-4503-3738-0.
DOI: http://dx.doi.org/10.1145/0000000.0000000

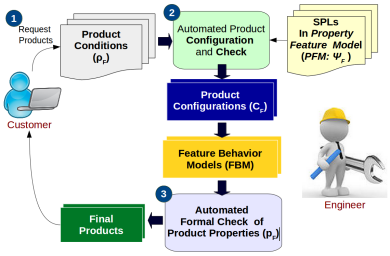

Figure 1: Our formal SPLE framework

The overall challenge is to analyze a family of real-time systems (rather than a single one), depending on varying sets of resources. Two main issues are raised for the verification of an SPL of real-time systems. 1) The specification method must link individual features of an SPL to the corresponding real-time properties that must be verified. 2) The analysis method must verify all products generated from an SPL against all real-time properties imposed upon individual features of each product. If the products of an SPL are safety-critical, this analysis method should be rigorous enough to guarantee the safety of all the products.

This paper proposes a formal SPLE framework for real-time scheduling units[1] and demonstrates its efficiency and feasibility. It focuses on the formal analysis of real-time properties of an SPL in terms of resource sharing with time dependent functionalities. Our framework is depicted in Figure 1. It provides a structural description method of the variability and the properties of a real time system, and behavioral models to verify the properties using formal techniques and the tools UPPAAL symbolic model checker (MC) [2] and UPPAAL statistical model checker (SMC) [6].

The rest of the paper is organized as follows: Section 2 presents a new extension of a feature model, Property Feature Model (PFM). In Section 3, we provide feature behavioral models of the components of a scheduling unit. In Section 4, we present the results from a case study. Finally Section 5 discusses related work and Section 6 concludes this paper.

[1]A scheduling unit consists of tasks and a scheduling mechanism.

2. PROPERTY FEATURE MODEL

We analyze SPLs of real-time systems with respect to the following 3 properties: deadlock-freedom, schedulability, and performance. Inspired by [8,10], we propose a new extension of feature model called Property Feature Model (PFM) that distinguishes between features and properties using property-specific operators. It states two pieces of important information: the scope of a property and a list of properties that individual features must satisfy.

Syntax for PFM. A PFM is described using the similar notations of a FM. The root of a PFM is a feature, that has child features or properties. A property node can have another property node as its child, but not a feature node. A property can be represented by the composition of multiple properties.

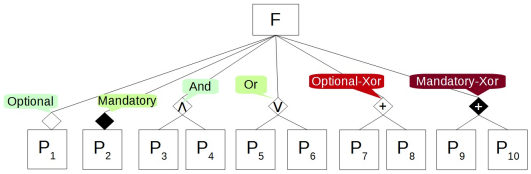

Figure 2: Property-specific operators PFM

Figure 2 shows property-specific operators of a PFM in graphical notations. Similar to the optional feature of a feature model, an optional property of a PFM can represent two products: one that satisfies the property and one that does not. Feature and property nodes can be quantified or given parameters for their products.

Figure 3 shows an example of PFM. The SPL has the root feature SS representing a scheduling unit. It is composed of two mandatory features Task1 and CPU1, one optional feature Task2, and one mandatory property "Not Deadlock." The feature CPU1 is mandatory and quantified by two schedulers, FP (Fixed-Priority) or EDF (Earliest Deadline First). The property node denoted by Not Deadlock states a global property that requires that the root feature SS is never in deadlock when it operates. The property node denoted by Schedulable imposed upon Task2 is a mandatory and local property specifying that Task2 can never miss its deadline.

A PFM can be represented by a propositional logic formula with Boolean variables [3]. Each Boolean variable corresponds to a single feature f stating whether the feature is included or not or the satisfiability relation $f \models p$. We allow numeric and arrayed features in propositional logic formulas, like F[A, B, C], instead of Boolean variables [9].

Definition 1. *$PFM = \{\mathcal{F}, \mathcal{P}, \rightarrow, \models, \psi_F\}$ such that*

- *$\mathcal{F} = \{f_0, ..., f_n\}$ is a set of features, f_0 being the root feature,*
- *$\mathcal{P} = \{p_1, ..., p_m\}$ is a set of properties,*
- *$\rightarrow \in 2^{\mathcal{F}}$ is a parent to child feature relation that encodes the feature structure of the PFM,*
- *$\models \in \mathcal{F} \times \mathcal{P}$ is a satisfiability relation ($f \models p$) meaning that a feature f satisfies a property p.*
- *ψ_F is a propositional logic formula over features and properties that represents the constraints of the PFM.*

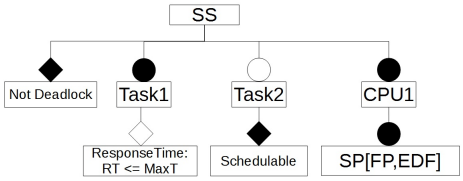

Figure 3: An SPL of a scheduling unit

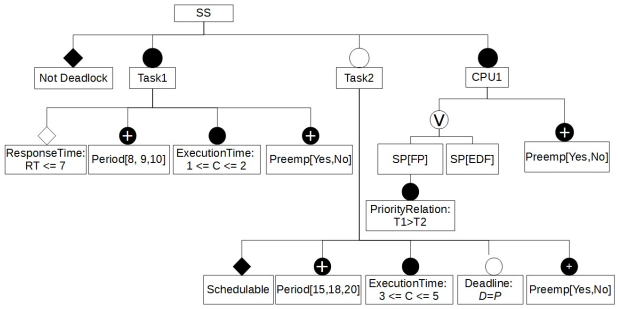

Figure 4: The refined PFM of a scheduling unit

Notice that ψ_F includes both a relation between parent and child features and a relation between features that are not in the parent-child relation. In the case where a feature is in association with another feature that is neither parent nor child, an additional proposition logic formula is given to define such a relation.

Product Configuration. A product generated from a PFM is set of included features that satisfy the constraints of the PFM. We define a product condition that is used to describe requirements of product features requested by the customer. A *product condition* ρ_F is a propositional formula that is a conjunction of condition variables corresponding to individual features in $\{f_0, f_1, ..., fn\}$. It is defined by the following grammar:

$$\begin{aligned} \rho_F &::= c \mid \neg c \mid \rho_F \wedge \rho_F \mid e \\ e &::= x == d \mid x > d \mid x < d \mid x \leqslant d \mid x \geqslant d \mid x[f] \\ f &::= d, f \mid d \end{aligned}$$

where c is a Boolean variable, x is a numeric or constant variable that are not allowed to contain negation, and d is a numeric or constant value. A product condition is checked against the PFM to see if the proposed products are producible from the PFM specified by the product condition. This check can be performed by SMT-solvers [5, 7, 9].

To derive products from a PFM, we define a (product) *configuration* that is a set of condition variables that imply the inclusion, exclusion, or valuation of the corresponding features. Compared to a product condition, it is used to generate all possible products of an SPL, which should satisfy all product conditions that customers require.

Definition 2. *(Configuration): A* configuration *γ is a set of condition variables $c_i \in \{true, false, v\}$, each corresponding to a feature $f_i \in \mathcal{F}$ or a property $p_i \in \mathcal{P}$, such that*

- *$c_i = true$ represents the inclusion of f_i or p_i,*
- *$c_i = false$ represents the exclusion of f_i or p_i,*
- *$c_i = v$ represents the assignment of f_i to a value v in any type.*

For a given PFM, a configuration of a product is created by assigning c_i to one of *true*, *false* or a value v, where c_i has a corresponding feature or property in the PFM. A configuration γ is "*determined*" if no variable c_i remains undetermined, i.e. not included in γ. Then $|\gamma|$ is equal to $|\mathcal{F}| + |\mathcal{P}|$.

Definition 3. *(Propositional Logic Formula Projection): The projection of ψ_F over a configuration γ, denoted by $\psi_F|_\gamma$, returns the formula ψ_F in which every variable v_i corresponding to a feature f_i or a property p_i has been substituted with the value of the corresponding condition variable c_i in γ [9].*

A configuration γ is said to be "*valid*" if $\psi_F|_\gamma$, holds, i.e. the configuration is producible from a feature model ψ_F. Otherwise, the configuration γ is said to be "*invalid*." Formally, a product is a *valid* and *determined* configuration.

Now, we define a non-deterministic decision process that allows to construct all the products of a PFM compatible with the product condition ρ_F expressed by the customer. The process starts from the configuration $\gamma_0 = \{c_0 = true\}$ that only includes the root feature of the PFM, and it recursively extends this configuration until all the features and all the properties have been determined. Therefore, from a configuration γ a new configuration γ' is produced by extending γ with a feature condition c_i, according to the following rules:

1. $\gamma' = \gamma \cup c_i$,
2. $\exists c_j \in \gamma$ such that either $f_j \rightarrow f_i$, which means that f_i is a child feature of f_j that has already been determined to be included, or $p_i \vDash f_j$, which means that p_i is a property of f_j already determined,
3. $\psi_F|_{\gamma'}$ and $\rho_F|_{\gamma'}$ hold.

The first rule produces a new configuration by including the condition variable c_i corresponding to the decision on the feature f_i or the property p_i. The second rule restricts the decision process to make it follow the order from parent to child defined in the PFM. The last rule checks if a new configuration (γ') satisfies both feature constraints (ψ_F) and customer's requests (ρ_F).

3. FEATURE BEHAVIORAL MODEL

A SPL of a scheduling unit is analyzed to see if the products generated from the SPL satisfy their properties. To this end, all products from an SPL are represented by behavioral models of real-time scheduling units. We model them using TA and SWA such that properties of an SPL can be analyzed using the behavioral models.

The scheduling units that we consider in this paper are preemptive, so that the execution of a task can be interrupted by other tasks according to a scheduling policy. Figure 5 shows the feature behavioral model of a real-time task with preemption, inspired by the work in [4]. We have extended this model with variables that encode the enabling, disabling or valuation of the features.

The SWA task model in Figure 5 is a generic model that can be configured to execute any configuration of task producible from the PFM. It captures the behavior of a task

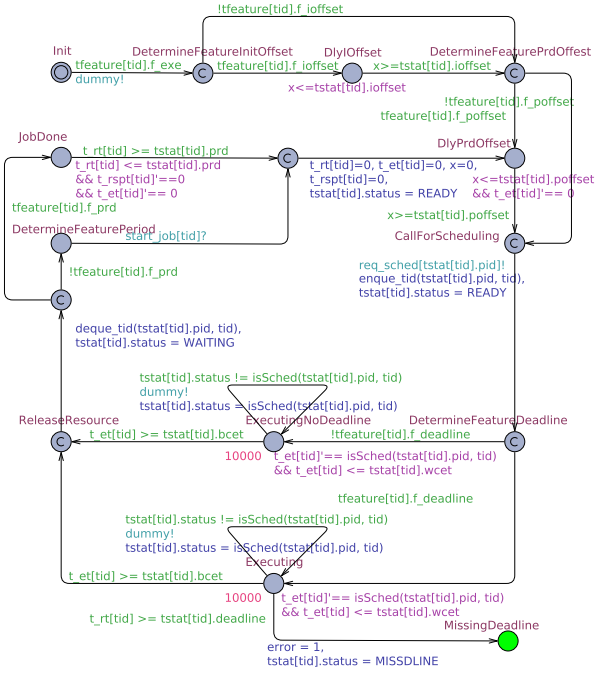

Figure 5: A SWA task model for a family of tasks

after the feature variables have been configured at initialization. For instance, the location DetermineFeatureInitOffset has two out-going transitions: one to DlyOffset, and the other to DetermineFeaturePrdOffset. The transitions are labeled with a guard that distinguishes a feature and the property of a task is determined by a set of enabled guards. Thus, the transition guard tfeature[tid].f_ioffset is set to true if the feature InitOffset is included, but set to false when the feature is excluded.

While executing, the task may be preempted by the resource scheduler. The preemption is implemented by a stopwatch clock t_et[tid] that can stop and resume [4]. It represents the remaining execution time of the task tid and it should progress only when the CPU resource is available to the task. This stopwatch t_et[tid] is constrained by an invariant that is associated with a function isSched().

4. EVALUATION

This section presents results of analyzing the SPL of Figure 4. Using UPPAAL MC we check the schedulability of the tasks and deadlock freedom as well. UPPAAL SMC is used to estimate the worst-case execution time of tasks, individually.

In addition to the feature behavioral models of tasks and resources, we provide a configuration template that generates configurations of real-time systems out of a given PFM before the execution of the system. A configuration template simulates the non-deterministic decision process presented in Section 2 and selects features from a PFM in a non-deterministic way to make a configuration of the system under analysis

The running example of Figure 4 has only 2 tasks and no constraints over configurations. The feature Task1 has 6 configurations, the feature Task2 has 12 configurations, and the

Table 1: Timing analysis results for the SPL in Figure 4

Feature	Query for Property	Results	Time
SS	A[] not deadlock	Yes	28.43s
$Task1$	E[<=10000;100](max:t_rspt[1])	6.80	3.22s
$Task2$	A[] (tstat[2].status ! = MISSDLINE)	Yes	29.85s

feature CPU1 has 4 configurations. SS has 24 configurations without Task2, and 288 configurations with Task2.
Table 1 shows the results of analyzing the properties included in the SPL. First, the property of SS "Not Deadlock" is formulated as the Computation Tree Logic (CTL) query "A[] not deadlock" stating that the system is deadlock-free. The property is proven to hold in the system. Second, the schedulability of Task2 is analyzed. The CTL query, "A[] (tstat[2].status ! = MISSDLINE)", is used as a specification, meaning that the state variable tstat[2].status can never be the same as "MISSDLINE" while the system is running. UPPAAL MC verified that Task2 never misses the deadline. Third, we analyzed the performance, i.e. the response time of a task, of configurations from the SPL. The property RT<=7 upon Task1 in the SPL is represented by a SMC query, E[<=10000;100](max:t_rspt[1]), requiring UPPAAL SMC to compute the average of the maximum value of t_rspt[1] for 10,000 simulation times by 100 simulation rounds.

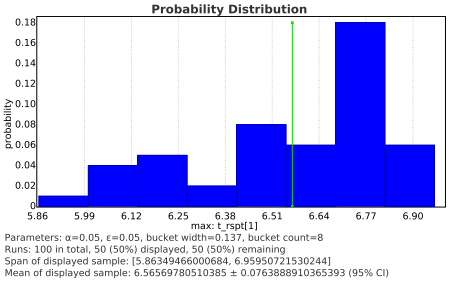

Figure 6: Probability distribution of Task1's response time

UPPAAL SMC produces a probability distribution, as the answer to the query, shown in Figure 6. It shows that the response times of the task is at most 6.80 time units during the simulation and validates that the worst-case response time of Task1 is less than 7. Table 2 shows the analysis results of another case-study containing 5 real-time tasks and 2 resource schedulers.

Table 2: Timing analysis results

Feature	Query for Property	Results	Time
SS	A[] not deadlock	Yes	25,322.8 s
$Task1$	E[<=10000;100](max:t_rspt[1])	66.6	0.28s
$Task4$	E[<=10000;100](max:t_rspt[4])	71.75	0.31s
SS	A[] not miss_deadline	Yes	25,134.51 s
SS	Pr [<=1000000] <> miss_deadline	[0,0.0199955]	302.51 s

5. RELATED WORK

The formalism of feature models (FM) in this paper relies on the basic and classical constructs of [1]. Our extension of FM was inspired by Kang et al. [8] that criticizes the existing FM by saying that it often specifies one or more concerns of SPL in one FM. Related to the quality of an SPL, they proposed an attribute-based FM where only qualities of products are separately given as a FM. However, such a representation makes it hard to explicitly figure out the relationship between a feature and the associated quality attributes (i.e. properties). For this reason, this paper extended FM with the related properties so that a verification property is associated to a feature in one FM through specific operators.

6. CONCLUSIONS

SPLE aims to provide efficient engineering solutions for building multiple products that share common features. This paper proposed a formal framework dedicated to the verification of SPLs that should satisfy schedulability properties. Specifically, we proposed a new formalism for variability modeling, called PFM, to define feature models together with feature properties, and defined the notion of product condition that represents customer's product requests. We formally defined the semantics of PFM so that the SPL modeled in the PFM can automatically generate valid configurations in compliance with customer's requests. In order to analyze the configured products against feature properties, we proposed behavioral models that capture the features of real-time scheduling units defined in the PFM. We then showed how a set of scheduling units in an SPL specification can be automatically verified against the set of required properties by leveraging efficient model checking methods. Throughout the paper we illustrated the formal framework with a family of scheduling units and showed the applicability and efficiency of our techniques.

As future work we plan to investigate the scalability of our proposal w.r.t. large, variability-intensive scheduling systems. We also want to include a wider range of schedulability properties in our verification process.

7. REFERENCES

[1] M. Acher, P. Collet, P. Lahire, and R. B. France. Familiar: A domain-specific language for large scale management of feature models. *SCP*, 78(6), 2013.

[2] G. Behrmann, A. David, K. G. Larsen, J. Håakansson, P. Pettersson, W. Yi, and M. Hendriks. UPPAAL 4.0. In *Proc. of QEST*, pages 125–126, 2006.

[3] D. Benavides, S. Segura, and A. Ruiz-Cortes. Automated analysis of feature models 20 years later: a literature review. *Information Systems*, 35(6), 2010.

[4] A. Boudjadar, A. David, J. H. Kim, K. G. Larsen, M. Mikucionis, U. Nyman, and A. Skou. Hierarchical scheduling framework based on compositional analysis using uppaal. In *Proc. of FACS*, pages 61–78, 2013.

[5] M. Cordy, P. Schobbens, P. Heymans, and A. Legay. Beyond boolean product-line model checking: dealing with feature attributes and multi-features. In *Proc. of ICSE*, pages 472–481, 2013.

[6] A. David, K. Larsen, A. Legay, M. Mikučionis, and D. Poulsen. UPPAAL SMC tutorial. *STTT*, 17(4):1–19, 2015.

[7] V. Ganesh. *Decision Procedures for Bit-vectors, Arrays and Integers*. PhD thesis, Stanford, CA, USA, 2007.

[8] K. Kang and H. Lee. Variability modeling. In *Syst. and Software Variability Manage.*, pages 25–42. 2013.

[9] H. Sabouri, M. Jaghoori, F. de Boer, and R. Khosravi. Scheduling and analysis of real-time software families. In *Proc. of COMPSAC*, pages 680–689, July 2012.

[10] 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.*, 47(1):6:1–6:45, June 2014.