Automated Model-based Performance Analysis of Software Product Lines under Uncertainty

Paolo Arcaini

National Institute of Informatics, Japan

Omar Inverso

Gran Sasso Science Institute, Italy

Catia Trubiani

Gran Sasso Science Institute, Italy

Abstract

Context: A Software Product Line (SPL) can express the variability of a system through the specification of configuration options. Evaluating performance characteristics, such as the system response time and resource utilization, of a software product is challenging, even more so in the presence of uncertain values of the attributes.

Objective: The goal of this paper is to automate the generation of performance models for software products derived from the feature model by selection heuristics. We aim at obtaining model-based predictive results to quantify the correlation between the features, along with their uncertainties, and the system performance. This way, software engineers can be informed on the performance characteristics before implementing the system.

Method: We propose a tool-supported framework that, starting from a feature model annotated with performance-related characteristics, derives Queueing Network (QN) performance models for all the products of the SPL. Model-based performance analysis is carried out on the models obtained by selecting the products that show the maximum and minimum performance-based costs.

Results: We applied our approach to almost seven thousand feature models including more than one hundred and seventy features. The generation of QN

models is automatically performed in much less than one second, whereas their model-based performance analysis embeds simulation delays and requires about six minutes on average.

Conclusion: The experimental results confirm that our approach can be effective on a variety of systems for which software engineers may be provided with early insights on the system performance in reasonably short times. Software engineers are supported in the task of understanding the performance bounds that may encounter when (de)selecting different configuration options, along with their uncertainties.

Keywords: Software Product Lines, Software Performance Engineering, Attributed Feature Models, Queueing Networks, Uncertainty

1. Introduction

Software Product Line (SPL) engineering enables the specification of software systems sharing some common characteristics in terms of configuration options, i.e., *features* [1, 2]. System features allow to specify design alternatives early in the development process [3], in order to characterize a portfolio of similar products. However, the engineering of SPLs is an inherently complex process where different stakeholders (e.g., managers, software architects, software developers, etc.) may adopt conflicting choices. Therefore, converging towards a system design (in terms of a *feature model* [4]) can be challenging, even more so when features have attributes with values (i.e., an *attributed feature model* [5]) that are not fixed but span intervals of values [6], and are possibly subject to quantitive constraints [7].

Recently, there has been a growing interest in variability modeling and analysis techniques that do explicitly consider quantitative (i.e., non-functional) requirements, such as dependability, energy consumption, security, cost, etc. [8]. Among them, performance is indeed rather critical, because it directly affects user satisfaction. Moreover, since program fixes can cause performance fluctuations, the relationship between what a system does and how fast it works is

deeper than it would appear at first, to the extent that performance has been recently rethought of as the *new correctness* [9].

As possible examples of performance requirements, one may want to limit the system response time, guarantee a minimum service throughput, enforce fairness constraints on the utilization of hardware devices, and so on. If performance targets are not met, a variety of negative consequences arise (e.g., damaged customer relations, economic loss, etc.), and may lead to expensive rework. This motivates predictive techniques for an early quantitative evaluation of system performance at development time [10, 11].

A methodology for integrating model-based performance analysis in the software development process was proposed in [12]. The key idea is to complement the feature model with performance-related abstractions (i.e., the UML MARTE profile [13] is used for specifying resources characteristics, e.g., the *service times*), and then taking them into account to generate *performance models* for specific software products.

A widely applied formalism for modeling performance, especially in resource-sharing contexts [14, 15, 16, 17], are *Queueing Networks* (*QNs*) [18, 19]. In previous work [20], following the procedure defined in [12], we used QNs to model the performance of multiple system variants generated from a given feature model, so to quantitatively compare them. Our approach, however, suffered from two main limitations: (i) manual encoding of performance-annotated feature models to QNs, which is time-consuming and prone to errors; (ii) arbitrary assumptions on the compositionality (i.e., sequential or parallel) of different system functionalities within the same feature group.

In this paper, we address the above limitations by automating the process of transforming feature models into Fork-join QNs [18, 19]. We allow to annotate the feature model with performance-based indications on how different feature groups are composed, and on uncertainty in the attributes values (e.g., the service time). We present a tool that implements mapping rules to build Fork-join QNs out of the annotated feature models. Note that the feature model is a static representation of the system, while the QN model provides a dynamic

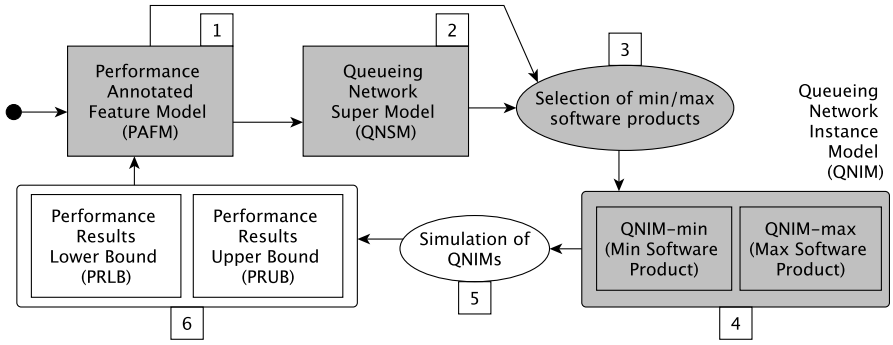


Figure 1: Overview of the proposed approach.

perspective. The assumption to get from the static to the dynamic perspective is to exploit the performance-based annotations specified as part of feature models, in order to derive a meaningful specification of the corresponding queueing network.

The overall workflow is outlined in Figure 1, where shaded boxes highlight the main novelties. We developed a tool-supported framework that takes as input a *Performance Annotated Feature Model (PAFM)*, i.e., an extension of attributed feature models [5], where features are annotated with performance-related characteristics (e.g., the service time) and feature groups indicate how to compose the individual functionalities. Such model is automatically transformed into a comprehensive performance model that characterizes all the products of the SPL, i.e., the *Queueing Network Super Model (QNSM)*. The selection of features from PAFM leads to a software product that is used to derive an instance model out of QNSM, i.e., the *Queueing Network Instance Model (QNIM)*. In general, software products can be selected by the user or by applying specific heuristics; our approach uses FAMA [5] to automatically generate the two software products showing minimum and maximum performance-based costs. From these two products, two queueing networks (i.e., *QNIM-min* and *QNIM-max*) are generated by applying our mapping rules and simulated with JMT [21] for performance evaluation. Performance analysis is carried out on QNIM-min and

QNIM-max (see Figure 1) that are performance models of two specific software products, thus our approach belongs to the product-based analysis category, according to the classification by Thüm et al. [22]. Performance measures of interest (such as system response time, resource utilization, or service throughput) are then reported in terms of lower and upper bounds denoting the uncertainty in the values of features' attributes. The design may be refined and the whole cycle re-executed again in case any performance requirement is not fulfilled. Our approach can provide the designers with early insights about the correlation between the selection of features (along with their uncertainty) and the system performance. We experimented on 6934 models including up to 176 features, for which we could obtain model-based performance analysis in about 6 minutes on average.

The rest of the paper is organized as follows. Section 2 provides some background information. Section 3 illustrates the proposed approach: Section 3.1 presents PAFMs, Section 3.2 describes the mapping rules to automate the transformation from PAFM into QNSM, and Section 3.3 argues on how to instantiate a QNIM from a QNSM starting from a specific product. Section 4 provides a quantitative evaluation of the proposed approach. Section 5 discusses possible threats to the validity. Section 6 reports relevant related work. Section 7 concludes the paper and provides future research directions.

2. Background

In this section we provide some background information on Feature Models (FMs) [4] and Queueing Networks (QNs) [19]. We also introduce some basic definitions that we use afterwards.

2.1. Feature Models

Feature models are commonly used as a compact representation of all the products in a Software Product Line (SPL) [23]. A feature model is graphically represented as a tree-like structure in which nodes represent features, and con-

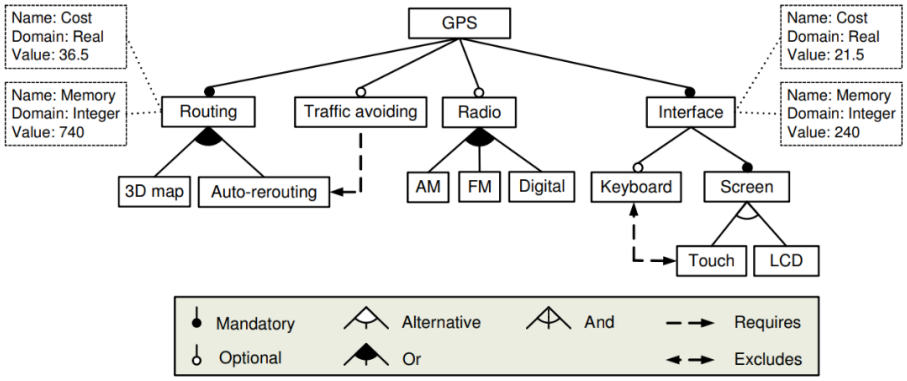


Figure 2: Feature Model of a GPS system (adapted from [25]).

nections illustrate the relationships between them. These relationships establish how features can be combined to form valid products [24].

Figure 2 reports an example of a feature model representing Global Position System (GPS) devices. We exploit this example to introduce the syntax and semantics of feature models. Each feature model has a *root* feature that identifies the SPL. The root feature of the example is GPS. Among the features of the feature model, it is possible to specify the following constraints [4, 26]:

- **Mandatory**: if a feature has a mandatory relationship with its parent feature, it must be included in all the products in which its parent feature appears. For example, in Figure 2, all products must provide support for *Routing* and *Interface*;

- **Optional**: if a feature has an optional relationship with its parent feature, it can be optionally included in products that include its parent feature. For instance, in Figure 2, *Keyboard* is defined as an optional feature of the *Interface*;

- **AND**: a feature and its group of optional and/or mandatory children constitute an *AND* group (e.g., *GPS* and its children is an AND group, and *Interface* and its children is another AND group). To simplify the translation from FM to QN (see Section 3.2) we also consider each child

as an AND group on its own;

- **Alternative**: a set of child features are defined as *alternative* if exactly one feature must be selected when its parent feature is part of the product. For example, in Figure 2, each product must provide support for either a *Touch* or an *LCD Screen*, but not both;

- **OR**: child features are said to have an *or* relation with their parent when one or more of them can be included in the products in which its parent feature appears. For instance, in Figure 2, each product must provide support for at least *3D map* viewing or *Auto-rerouting*, or both of them.

In addition to the parental relationships between features, a feature model can also contain *cross-tree constraints* between features that are:

- **requires**: if a feature a requires a feature b, the inclusion of a in a product implies the inclusion of b in this product. For example, in Figure 2, devices with *Traffic avoiding* require the *Auto-rerouting* feature.

- **excludes**: if a feature a excludes a feature b, both features cannot be part of the same product. For instance, in Figure 2, devices with *Touch Screen* exclude the support for a *Keyboard*.

Given a feature a, we will indicate with A the subtree of the feature model having a as root (not considering cross-tree constraints). Moreover, we will indicate with $children(a)$ all the children of a feature a.

Definition 1 (Configuration)**.** Given a feature model *fm* defined over a set of features F, a *configuration* $p = \{f_1, \ldots, f_m\}$ is a subset of F (i.e., $p \subseteq F$).

Definition 2 (Product)**.** A *product* is a configuration including a selection of features that respects all the constraints of the feature model.

In this work, we consider *attributed feature models* [27], i.e., an extension of feature models that allows to specify attributes (e.g., service time, memory consumption, cost, etc.) over features. Note that an attributed feature model

can include multiple attributes. For example, feature *Routing* in Figure 2 is associated to two attributes: *cost* whose value is 36.5, and *memory* whose value is 740. Attributes may also be specified through an interval of values (representing their uncertainty) and this leads to the following definitions.

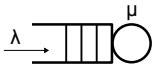
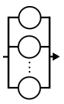
Definition 3 (Attributed feature model)**.** An *attributed feature model* $\overline{fm}$ is a feature model fm whose features $\overline{F}$ are annotated by attributes (e.g., a, b, c), i.e., $\overline{F} = \{(f_1, [a_1^L, a_1^U], [b_1^L, b_1^U], [c_1^L, c_1^U]), \ldots, (f_n, [a_n^L, a_n^U], [b_n^L, b_n^U], [c_n^L, c_n^U])\}$. An attribute a, when referring to a feature f_x, is defined as an interval of values (due to its intrinsic uncertainty), in the form $[a_x^L, a_x^U]$, where a_x^L and a_x^U represent the lower (L) and upper (U) *bounds* of the interval, respectively. $\overline{fm}$ can also contain constraints among the attributes of the features.

In the sequel of the paper, for the sake of readability, all the definitions are provided considering one attribute only. However, they can be extended to multiple attributes.

Definition 4 (Weighted configuration and weighted product)**.** Given a configuration $p = \{f_1, \ldots, f_m\}$ for an attributed feature model $\overline{fm}$ with attribute a, a *reward assignment* $r = \{r_1, \ldots, r_m\}$ is an assignment of reward values (i.e., $r_i \in [a_i^L, a_i^U]$ is the reward for the feature f_i). The purpose of reward assignments is to associate estimations to attributes for every feature in the configuration. Further details on how to calculate rewards are reported in Section 3.1. A *weighted configuration* is defined as $w = \{(f_1, r_1), \ldots, (f_m, r_m)\}$. We identify the set of features of a weighted configuration as $features(w) = \cup_{(f_i, r_i) \in w} \{f_i\}$. A *weighted product* is a weighted configuration that respects all the constraints of $\overline{fm}$ (on features and on attributes).

The FAMA framework [5] allows to model and analyze attributed feature models. For example, it allows to retrieve the products having the maximal and minimal sum of a given attribute. At the time of writing, the latest version of FeatureIDE [28] allows to specify feature attributes, but it does not allow to do any analysis. For this reason, we use FAMA to compute the minimal

Table 1: Main elements constituting Fork-join QN models.

Type of Station	Graphical representation	Description
Service Center		resources managing a load of λ rate and serving requests with a μ rate
Delay		customers are delayed by the defined station service time
Source		used to model open workloads whose interarrival time is defined as a rate
Sink		used to model customers leaving the system
Fork		splits the jobs into several tasks that are executed in parallel
Join		used to recombine the tasks a job had been previously split
Routing		jobs are routed to multiple stations by means of associated probabilities

and maximal system products while considering the performance attribute of interest.

2.2. Fork-join Queuing Networks

Queuing Networks (QNs) have been widely applied to represent and analyze resource sharing systems [19]. Table 1 schematically lists the main elements of Fork-join QN models. Specifically, a QN model is a collection of interacting *service centers* representing system resources and a set of *jobs* representing the users sharing the resources. Service centers model system resources that process customer requests. Each service center is composed of a *Server* and a *Queue*. Queues can be characterized by a finite or an infinite length. Service centers are

connected through *links* that form the network topology. Each server, contained in every service center, picks the next job from its queue (if not empty), processes it, and selects one link that acts as a *routing* station, i.e., it routes the processed request to the queue of another service center.

The time spent in every server by each request is modeled by exponential distributions. Incoming workload can be modeled as open (i.e., specified by an arrival rate λ) or closed (i.e., a constant number of jobs is specified as the population size). In case of open workload, jobs are generated by *source* nodes connected with links to the rest of the QN, and terminate in *sink* nodes when all tasks have been performed. *Delay* centers are nodes of the QN similar to service centers, but they do not have an associated queue. These centers are described only by a service time that denotes how long jobs are delayed before proceeding in further delay or queueing centers. In other words, the QN representation is a direct graph whose nodes are service centers and their connections are represented by the graph edges. Jobs go through the graph's edge set on the basis of the behavior of customers' service requests. Moreover, *fork* nodes are used to express parallelism, i.e., one task is split in multiple activities that are executed in parallel and synchronized through the *join* node. A routing node routes jobs to different branches $b_1, \ldots, b_n$ according to some given probabilities $p_1, \ldots, p_n$, with $p_i \in [0, 1]$ and $\sum_{i=1}^{n} p_i = 1$.

3. Proposed Approach

In this section we illustrate the core contribution of the paper. Specifically, we describe the Performance Annotated Feature Model (see Section 3.1), define the mapping rules to automate the transformation from PAFM into QNSM (see Section 3.2), and explain how to instantiate a QNIM from a QNSM for a specific product (see Section 3.3). The novelty of our work with respect to state-of-the art techniques [29] mainly lies in transforming feature models in analytical performance models, and exploiting the uncertainty in the attributes to derive lower and upper bounds for performance characteristics (e.g., system

response time) of interest.

3.1. Performance Annotated Feature Model

To derive a QN performance model from an attribute feature model, we first need to annotate some performance-based characteristics that enable the process of model-based performance evaluation. To this aim, we use the *ServiceTime* attribute (*st*). However, classical attributed feature models are not expressive enough for our purposes. In fact, while the semantics of a single feature is clear (i.e., each feature represents a functionality that has a given service time and can be mapped in a QN service center), interpreting the behavioral pattern of a group of features (i.e., *OR* or *AND*) to compute the global reward of a software product is more difficult.

For example, FAMA assumes that all the features selected in a product contribute to the final reward, but, when dealing with service times, different semantics can be devised on the basis of the considered functionalities. Therefore, we propose an extension of attribute feature models, *Performance Annotated Feature Model* (PAFM) – see the box labeled as 1 in Figure 1, in which we allow to specify two different reward semantics for *OR* and *AND* groups, *parallel* (P) or *sequential* (S). The groups are consequently named as AND^P, OR^P, AND^S, and OR^S. Both semantics can be used in the same feature model. Figure 3 depicts an example of PAFM suitable to represent a GPS system (adapted from [25]).

Definition 5 (Configuration reward)**.** Given a PAFM $\overline{fm}$, the *reward* of a weighted configuration $w = \{(f_1, r_1), \ldots, (f_m, r_m)\}$ is defined as:

$$reward(w) = reward(f^r, w)$$

being f^r root of $\overline{fm}$ (recall that the root feature is present in each valid config-

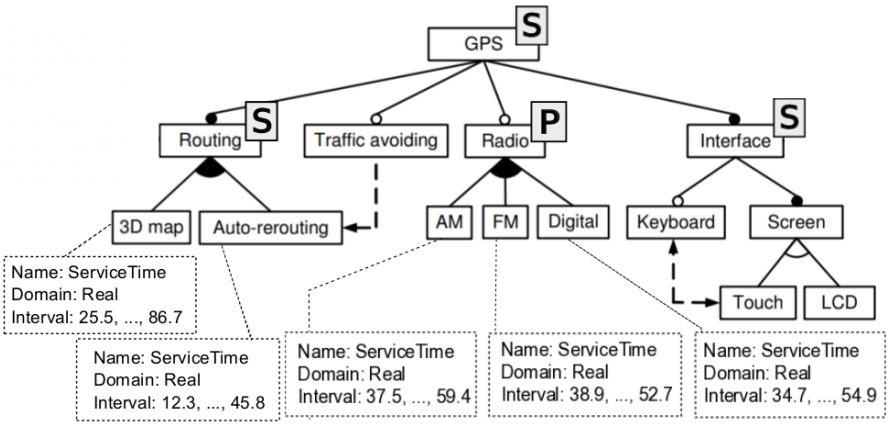


Figure 3: PAFM of a GPS system (adapted from [25]).

uration). The reward of a feature selected in w is inductively defined as

$$reward(f_i, w) = \begin{cases} r_i + \max\limits_{f \in CH(f_i,w)} reward(f, w) & \text{if } f_i \text{ is parent of } AND^P,\ OR^P, \text{ or } ALT \\ r_i + \sum\limits_{f \in CH(f_i,w)} reward(f, w) & \text{if } f_i \text{ is parent of } AND^S,\ OR^S \\ r_i & \text{if } f_i \text{ is a leaf} \end{cases}$$

being $CH(f_i, w) = \{f \in children(f_i) \mid f \in features(w)\}$, i.e., the children of f_i selected in w. Note that the proposed reward calculation entails that features whose attributes do not require a cumulative reward are handled by AND^P, whereas the cumulative case is modeled with AND^S. Our analytical performance model includes the *ServiceTime* attribute, whose semantics implies that if features are executed in parallel, then the overall *ServiceTime* is given by the maximum value of all the involved features.

Definition 6 (Sequential Group)**.** In a *sequential group* all the selected child features are executed sequentially. This indicates that the service times of all child features contribute to the overall computation and the system performance of the product they belong to. For example, in Figure 3 the feature *Routing* shows an S denoting that the selected child features will be sequentially executed. In case both *3D map* and *Auto-rerouting* will be selected, then the

reward for this group of features will be given by the sum of their values. Given the uncertainty in such values, the reward will span from 37.8 ($= 25.5 + 12.3$) as lower bound to 132.5 ($= 86.7 + 45.8$) as upper bound.

Definition 7 (Parallel Group)**.** In a *parallel group* all the selected child features are executed in parallel. This means that only the lowest and highest service times among all the child features contribute to the computation of the system performance of the product. Figure 3 shows an example of the feature *Radio* with *P*, and in case the three child features (i.e., *AM*, *FM*, and *Digital*) will be all selected. Due to the uncertainty in the values of these attributes, the reward for this group of features will span from 34.7 (i.e., the lowest value among the lower bounds) to 59.4 (i.e., the highest value among the upper bounds).

3.2. Queueing Network Super Model

In this section, we describe the mapping rules to transform a PAFM $\overline{fm}$ in a *Queueing Network Super Model* (QNSM) – see the box labeled as 2 in Figure 1.

Definition 8 (Queueing Network Super Model)**.** A QNSM consists of a queueing network *qn* and a set of constraints *Constr* among the branch probabilities of *qn*. A QNSM is defined as a *super model* since it represents the whole SPL: any assignment of values (including zero) to the branch probabilities that respects the constraints *Constr* identifies a valid product. Zero values in branches denote the exclusion of the corresponding features, as expected in some of the software products that can be generated from the QNSM.

For each feature of $\overline{fm}$, we generate a service center in *qn*. The network structure (i.e., the way service centers are connected) is derived from the feature model and the semantics of *AND* and *OR* groups. The basic parent-child relations are captured by the order of queues in the network: if a feature b is a descendant of a feature a in $\overline{fm}$, the service center q_b follows q_a in *qn* (being q_a and q_b the service centers describing f_a and f_b). The order of queues is also sufficient to model the semantics of AND^S.

The semantics of feature constraints (*Optional*, *Alternative*, OR^P, OR^S, and AND^P) is instead captured by a combination of QN components and constraints on branch probabilities of routing and fork components. In QNs, a branch probability describes the likelihood that the corresponding service center (and the subnetwork originating from it) is executed. However, in our setting, a feature is either selected or not selected: therefore, the corresponding queue is executed either always or never. For this reason, we require all the branch probabilities to be 0 or 1. Given a branch from A to B in a fork or routing group, there is a constraint in *Constr* so that $P(A, B) \in \{0, 1\}$.

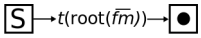
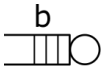
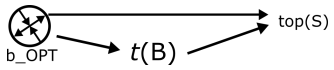
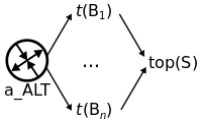
The transformation process from PAFM to QNSM is as follows. A `source` and a `sink` are generated as the starting and ending points of qn. A recursive process starts from the `root` node of $\overline{fm}$ and visits all its nodes. We identify with t the mapping function that is defined for each feature model element in terms of *mapping rules*.

During the mapping process, in order to model the parallel semantics, some elements e in the generated network qn must have a *synchronization node* `dest` where they synchronize their work with their siblings. This can be either the `sink` node of the network or a `join` node connected to the last opened `fork` node. For this reason, we keep a stack S of destination nodes. At the beginning, the stack only contains the `sink` node; when a `fork` is created, its corresponding `join` is added to the *top* of the stack. In qn, the `join` is linked to the element present on the top of stack. This guarantees that the forks are closed by the joins in the reverse order (i.e., the first fork matches the last join).

The mapping function t is initially applied to the whole feature model $\overline{fm}$. We identify two types of mapping rules: those that do not depend on the reward semantics of the FM element to translate, and those that do. Table 2 shows the mapping rules for the former category:

- the visit of $\overline{fm}$ consists in the generation of the initial `source` and the final `sink` nodes; the root of $\overline{fm}$ is recursively visited and the result of the visit is linked between the `source` and the `sink`;

Table 2: Mapping function t to transform some FM elements into QN elements.

FM element	QN element
$\overline{fm}$	
b	
b	t(B)
b	 $P(\texttt{b\_OPT}, \texttt{t(B)}) + P(\texttt{b\_OPT}, \texttt{top(S)}) = 1$
a $B_1 \ \ldots \ B_n$	 $\sum_{i=1}^{n} P(\texttt{a\_ALT}, \texttt{t(B}_\texttt{i}\texttt{)}) = 1$

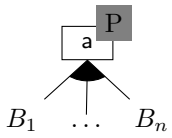
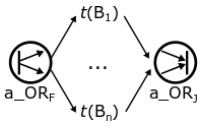
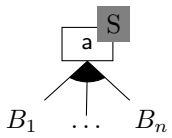
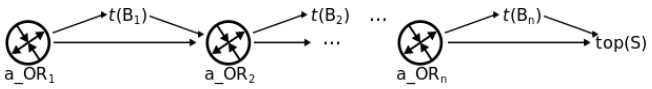
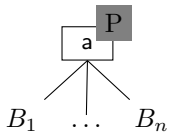
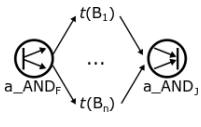
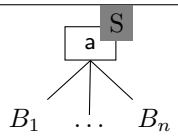
- a feature b becomes a service center. Its service time distribution $f(x) = \lambda_b e^{-\lambda_b x}$ is left parametric in terms of λ_b and its numerical value will be assigned during the product instantiation (see Section 3.3). Note that λ_b is related to the attribute st (modeling the service time) of the corresponding feature b. We recall that values of attributes are uncertain, hence the bounds of feature b depend on the intervals of st in b, i.e., $\lambda \in [\frac{1}{st_b^U}, \frac{1}{st_b^L}]$ (see Def. 3);

- given a mandatory feature b, the tree rooted in b (i.e., B) is visited;

- an optional feature b is modeled by a router `b_OPT` with two children, i.e., the tree B rooted in b (as for the mandatory feature) and the top of the stack `top(S)`. The router can only select one of the two children; therefore, the constraint on the router probabilities states that exactly one child is executed (the sum of branch probabilities is equal to 1).

- given an alternative group with parent a and children $b_1, \ldots, b_n$, a router

Table 3: Mapping function t to transform *OR* and *AND* groups into QN elements.

FM element	QN element
	 $\sum_{i=1}^{n} P(\texttt{a\_OR}_\texttt{F}, \texttt{t}(\texttt{B}_\texttt{i})) \geq 1$
	 $\bigwedge_{i\in\{1,\ldots,n-1\}} P(\texttt{a\_OR}_\texttt{i}, \texttt{t}(\texttt{B}_\texttt{i})) + P(\texttt{a\_OR}_\texttt{i}, \texttt{a\_OR}_{\texttt{i+1}}) = 1$ $\wedge \quad P(\texttt{a\_OR}_\texttt{n}, \texttt{t}(\texttt{B}_\texttt{n})) + P(\texttt{a\_OR}_\texttt{n}, \texttt{top(S)}) = 1$ $\sum_{i=1}^{n} P(\texttt{a\_OR}_\texttt{i}, \texttt{t}(\texttt{B}_\texttt{i})) \geq 1$
	 $\bigwedge_{i\in\{1,\ldots,n\}} P(\texttt{a\_AND}_\texttt{F}, \texttt{t}(\texttt{B}_\texttt{i})) = 1$
	$t(\texttt{B}_1) \longrightarrow t(\texttt{B}_2) \rightarrow \cdots \rightarrow t(\texttt{B}_\texttt{n})$

`a_ALT` is generated. Then, each subtree B_i (having child b_i as root) is visited; the result of the visit $\texttt{t}(\texttt{B}_\texttt{i})$ is added as child of `a_ALT` and connected to the top of the stack `top(S)`. In an alternative group, exactly one child must be selected; therefore, a constraint imposes that the sum of branch probabilities of all the children is exactly 1.

Table 3 shows the mapping of FM elements that include parallel and sequential semantics:

- given an *OR* group with parent a and children $b_1, \ldots, b_n$:
  - for parallel semantics OR^P, a fork $\texttt{a\_OR}_\texttt{F}$ and a join $\texttt{a\_OR}_\texttt{J}$ are gen-

erated; the join is pushed onto the stack S. Then, each subtree B_i (having child b_i as root) is visited, and the result of the visit is added as child of $\texttt{a\_OR}_\texttt{F}$. In an OR, at least one child must be selected. To this aim, the constraint on $\texttt{a\_OR}_\texttt{F}$ imposes that the sum of the probabilities is at least 1. At the end of the visit, the join $\texttt{a\_OR}_\texttt{J}$ is removed from the top of the stack S. Note that arranging the children in a fork-join guarantees the parallel semantics, i.e., the contribution to the overall system reward is given by the most expensive selected child;

- for sequential semantics OR^S, a router $\texttt{a\_OR}_\texttt{i}$ is built for each child b_i; each router $\texttt{a\_OR}_\texttt{i}$ is connected to $\texttt{t}(\texttt{B}_\texttt{i})$ (i.e., the result of the visit of the tree rooted in b_i) and the next router $\texttt{a\_OR}_{\texttt{i}+1}$ if $i < n$, or to the top of the stack $\texttt{top}(\texttt{S})$ otherwise. Constraints impose that, in each router, only one of the two branches is executed and that at least one child of the OR^S is executed. Note that this sequential composition guarantees that all the selected children contribute to the overall system reward.

- given an AND group with parent a and children $b_1, \ldots, b_n$:

  - for parallel semantics AND^P, a fork $\texttt{a\_AND}_\texttt{F}$ and a join $\texttt{a\_AND}_\texttt{J}$ are generated; the join is added to the stack S. Then, each subtree B_i (having child b_i as root) is visited, and the result of the visit is added as child of $\texttt{a\_AND}_\texttt{F}$. In an AND group, all the children are selected, so the probabilities of all the children must be set to 1. At the end of the visit, the join $\texttt{a\_AND}_\texttt{J}$ is removed from the top of the stack S. As already observed for OR^P, also in this case, the fork-join construction guarantees the desired reward semantics;

  - for sequential semantics AND^S, we simply concatenate all the $\texttt{t}(\texttt{B}_\texttt{i})$; this guarantees that the contribution to the overall system reward is given by all the selected children.

To model cross-tree constraints, i.e., *requires* and *excludes*, we need to create proper constraints on the branch probabilities (i.e., no structural element is added to the QN to explicitly model them). Given a service center sc, let $B(sc)$ be the set of branches of the queueing network that must be taken in order to execute sc. Let sc_a and sc_b be the service centers corresponding to two features a and b of the FM. Cross-tree constraints are modeled as follows:

- given a *requires* constraint from a to b, the constraint
$$\left(\bigwedge_{(e_1,e_2)\in B(sc_a)} P(e_1,e_2) = 1 \right) \rightarrow \left(\bigwedge_{(e_1,e_2)\in B(sc_b)} P(e_1,e_2) = 1 \right)$$
is added to *Constr*. The constraint imposes that if sc_a is executed, then also sc_b must be executed.

- given an *excludes* constraint from a to b, the constraint
$$\left(\bigwedge_{(e_1,e_2)\in B(sc_a)} P(e_1,e_2) = 1 \right) \rightarrow \neg \left(\bigwedge_{(e_1,e_2)\in B(sc_b)} P(e_1,e_2) = 1 \right)$$
is added to *Constr*. The constraint imposes that if sc_a is executed, then sc_b must not be executed.

Following these mapping rules, Figure 4 shows the QNSM obtained from the PAFM shown in Figure 3 (constraints in *Constr* are not reported).

Theorem 1 (Correctness)**.** Each QNSM generated from a PAFM is a topologically valid QN.

Proof. By construction, each QNSM is a directed acyclic graph that starts with a source (having no incoming connections) and terminates with a sink (having no outgoing connections). Each element in the graph is part of at least one path that connects the source with the sink. Each fork is related to multiple elements (i.e., the degree of parallelism requested by the user) and there exists a corresponding join (i.e., the point where the executions of the tasks are synchronized). Each router is connected to at least two elements. This graph represents a topologically valid QN, since it can be obtained by applying the construction rules of QNs, as shown in Table 1. □

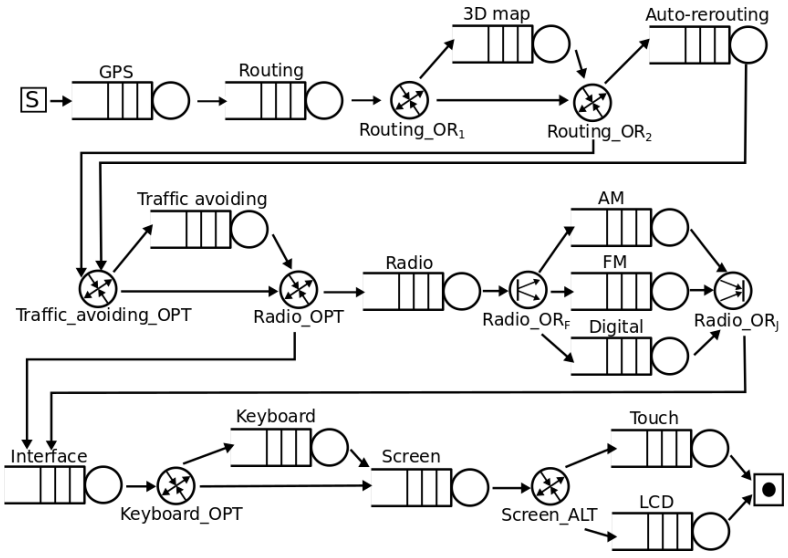


Figure 4: QNSM (automatically generated from the PAFM shown in Figure 3).

3.3. Queueing Network Instance Model

As seen in Section 3.2, the queuing network *qn* and the constraints *Constr* obtained through our mapping procedure constitute a *super model* QNSM, that represents *all* the possible products of the SPL described by PAFM. In this section, we introduce the *Queueing Network Instance Model* (QNIM) as a *qn* representing *one* particular product w of PAFM. A QNIM is formally defined as follows.

Definition 9 (Queueing Network Instance Model)**.** Given a QNSM, a QNIM represents a valid product given by any setting of the branch probabilities of QNSM (including values equal to zero that denote the exclusion of some features) that respects the constraints *Constr*.

The user can obtain a specific system product $w = \{(f_1, r_1), \ldots, (f_m, r_m)\}$ from PAFM, by either manually selecting it or using some tool as FAMA– see the box labeled as 3 in Figure 1 –. Using w, the user can instantiate it over the QNSM (i.e., *qn* and *Constr*). We propose two approaches to do so:

- Setting method: setting the branch probabilities of fork and routing groups

in qn such that only the service centers associated to the features of such specific product w are executed (Section 3.3.1);

- Slicing method: simplifying qn such that it contains only the service centers associated to the selected features of the specific product w (Section 3.3.2).

Note that the two queuing networks generated by these approaches are behaviorally equivalent, and produce the same model-based performance results. The only difference lies in the structure of the QN model, since slicing minimizes the number of queueing centers.

3.3.1. Setting method – Setting the feature costs and the routing probabilities

In order to instantiate a valid product[1] $w = \{(f_1, r_1), \ldots, (f_m, r_m)\}$ as a queuing network, we first need to set the rewards in the service centers associated to the selected features in w. If a feature f_i belongs to the product w, its service time distribution $st_i(x) = \lambda_i e^{-\lambda_i x}$ is set according to the feature reward r_i, namely $\lambda_i = \frac{1}{r_i}$; otherwise, the service time is set to zero.

We need to guarantee that only the service centers associated to the selected features are executed. This is obtained by initializing the fork and router probabilities visiting the queuing network from the source node and recursively applying these rules:

- given a service center q, the subnetwork originating from q is visited;
- given an AND^P fork $\texttt{a\_AND}_\texttt{F}$, all the branch probabilities $P(\texttt{a\_AND}_\texttt{F}, \texttt{t}(\texttt{B}_1))$, $\ldots$, $P(\texttt{a\_AND}_\texttt{F}, \texttt{t}(\texttt{B}_\texttt{n}))$ are set to one; then, all the children $\texttt{t}(\texttt{B}_1)$, $\ldots$, $\texttt{t}(\texttt{B}_\texttt{n})$ are visited;
- given an OR^P fork $\texttt{a\_OR}_\texttt{F}$, the branch probabilities $P(\texttt{a\_OR}_\texttt{F}, \texttt{t}(\texttt{B}_1))$, $\ldots$, $P(\texttt{a\_OR}_\texttt{F}, \texttt{t}(\texttt{B}_\texttt{n}))$ are set according to the product: for a given subnetwork

[1] Note that the configuration w is checked against the constraints *Constr* and, if w is not a valid system product, an exception is raised.

$\mathtt{t(B_i)}$, $P(\mathtt{a\_OR_F}, \mathtt{t(B_i)})$ is set to one if the feature b_i is present in the product, otherwise it is set to zero; then, all the subnetworks linked with probability equal to one are visited;

- given a router $\mathtt{r}$ (for optional feature, ALT, or OR^S) with branch probabilities $P(\mathtt{r}, \mathtt{t(B_1)}), \ldots, P(\mathtt{r}, \mathtt{t(B_n)})$, only the probability for the subnetwork $\mathtt{t(B_i)}$ (that corresponds to the selection of feature b_i in the product) is set to one, and all the other probabilities are set to zero; the selected subnetwork $\mathtt{t(B_i)}$ is then visited;

- given a join node $\mathtt{J}$ that has not been visited yet, the subnetwork originating from $\mathtt{J}$ is visited, and $\mathtt{J}$ is marked as *visited*.

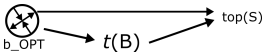
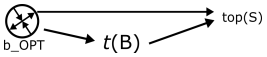
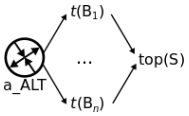
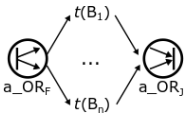
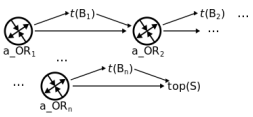
Note that the above rules allow to execute only the service centers corresponding to the selected features of the feature model. To complete the specification of the queuing network, we also need to set the branch probabilities of fork/routers that are not executed. Specifically, we give a default assignment that satisfies the constraints *Constr* (actually, any assignment would be fine as these branches are never reached in the simulation).

Table 4 shows the translation from QNSM (see QN elements in Tables 2 and 3) to QNIM. This transformation is automatically performed through the setting of probability values in routing stations, due to the corresponding product. For instance, the first line of Table 4 indicates that if $B \in F_w$, then the probability of routing is set to one, to include the node labeled $t(B)$ as part of the ones that need to be visited in QNIM. Table 4 shows the settings applied to the QNSM elements when considering a specific product under analysis, and this results in the set of QNIM elements that are required to be visited. For the sake of conciseness, we only report the elements that are affected by the translation (i.e., those having probabilities that must be set).

Theorem 2 (Soundness)**.** Each QN instantiated from a QNSM is *executable*, i.e., all the requests are delivered from the source to the sink.

Proof. We already proved in Theorem 1 that a QNSM is a topologically valid

Table 4: Translation from QNSM to QNIM – Setting method (w is a product and $F_w = features(w)$)

QN element in QNSM	Product w	Setting of P in QNIM
	$B \in F_w$	$P(\texttt{b\_OPT}, \texttt{t(B)}) := 1$ $P(\texttt{b\_OPT}, \texttt{top(S)}) := 0$
	$B \notin F_w$	$P(\texttt{b\_OPT}, \texttt{t(B)}) := 0$ $P(\texttt{b\_OPT}, \texttt{top(S)}) := 1$
	$B_i \in F_w$	$P(\texttt{a\_ALT}, \texttt{t(B}_\texttt{i}\texttt{)}) := 1$ $P(\texttt{a\_ALT}, \texttt{t(B}_\texttt{j}\texttt{)}) := 0$, with $j \neq i$
	$B_{i_1}, \ldots, B_{i_k} \in F_w$	$P(\texttt{a\_OR}_\texttt{F}, \texttt{t(B}_{\texttt{i}_1}\texttt{)}) := 1 \ldots$ $P(\texttt{a\_OR}_\texttt{F}, \texttt{t(B}_{\texttt{i}_\texttt{k}}\texttt{)}) := 1$ $P(\texttt{a\_OR}_\texttt{F}, \texttt{t(B}_\texttt{j}\texttt{)}) := 0$ with $j \notin \{i_1, \ldots, i_k\}$
	$B_{i_1}, \ldots, B_{i_k} \in F_w$	$P(\texttt{a\_OR}_{\texttt{i}_1}, \texttt{t(B}_{\texttt{i}_1}\texttt{)}) := 1 \ldots$ $P(\texttt{a\_OR}_{\texttt{i}_\texttt{k}}, \texttt{t(B}_{\texttt{i}_\texttt{k}}\texttt{)}) := 1$ $P(\texttt{a\_OR}_\texttt{j}, \texttt{t(B}_\texttt{j}\texttt{)}) := 0$ with $j \notin \{i_1, \ldots, i_k\}$ $P(\texttt{a\_OR}_{\texttt{i}_1}, \texttt{a\_OR}_{\texttt{i}_1+1}) := 0 \ldots$ $P(\texttt{a\_OR}_{\texttt{i}_\texttt{k}}, \texttt{a\_OR}_{\texttt{i}_\texttt{k}+1}) := 0$ $P(\texttt{a\_OR}_\texttt{j}, \texttt{a\_OR}_{\texttt{j}+1}) := 1$ with $j \notin \{i_1, \ldots, i_k\}$

QN, so there exists at least one path from the source to the sink. Such a path is guaranteed to be executable. In fact, if a feature is selected in the product, by construction, the corresponding queue is reachable, since all the probabilities leading to the queue are set to one. On the contrary, if an optional feature is not selected, the corresponding router determines a connection leading to the sink. This way, all the requests starting from the source are delivered to the sink, i.e., the QN is executable. □

3.3.2. Slicing method – Obtaining sliced queuing networks

The approach presented in Section 3.3.1 instantiates the QNSM for a particular product w, but also keeps in the network the service centers of features

that are not selected in w. This can be useful as the designer can, while testing a particular product, also reason on alternative products (as they would do on the feature model).

However, in some scenarios, the designer may prefer to see only the service centers (i.e., features) of a specific software product (for example, if the generated QN is large and difficult to handle). To this end, we also allow to obtain a *sliced queuing network* that only contains the service centers of the given product. Such network can be obtained by applying the following simplification rules to the network instantiated for a product (the one defined in Section 3.3.1):

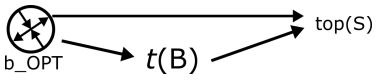
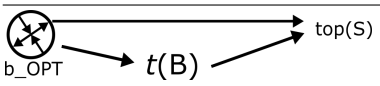
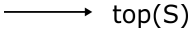
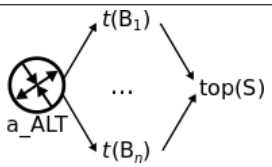
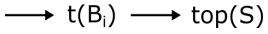
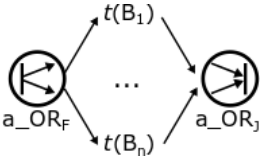
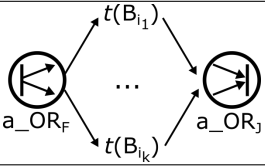
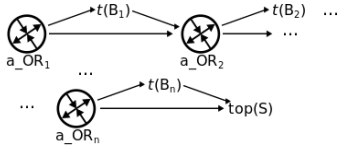
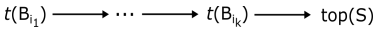
- given a fork `F` associated with a join `J` and subnetworks $\texttt{sn}_1$, ..., $\texttt{sn}_n$ between `F` and `J`, the subnetworks reached with branch probability equal to zero are removed, whereas the others are recursively simplified. After this process, if only one subnetwork $\texttt{sn}_i$ is left, `F` and `J` are removed and $\texttt{sn}_i$ inherits the father from `F` and the descendant from `J`. Instead, if no subnetwork is left, `F` and `J` are removed and the father of `F` becomes the father of the descendant of `J`.

- given a router `R` with subnetworks $\texttt{sn}_1$, ..., $\texttt{sn}_n$, `R` is removed and substituted with the only subnetwork $\texttt{sn}_i$ reached with probability equal to one (this is guaranteed to exist); then, $\texttt{sn}_i$ is recursively simplified.

Similarly to Table 4 (for the setting method), Table 5 shows the matching between QN elements in QNSM and their counterparts in QNIM when using the slicing method. For instance, the first line of Table 5 indicates that if $B \in F_w$, then the router is removed and the node labeled $t(B)$ is included as part of the ones that need to be visited in QNIM. For the sake of conciseness, we only report the elements affected by the translation.

As stated in Section 1, our approach relies on FAMA [5] to automatically produce the two software products showing maximum and minimum performance-based costs. From these two products, two queueing networks are instantiated (i.e., *QNIM-max* and *QNIM-min* – see the box labeled as 4 in Figure 1).

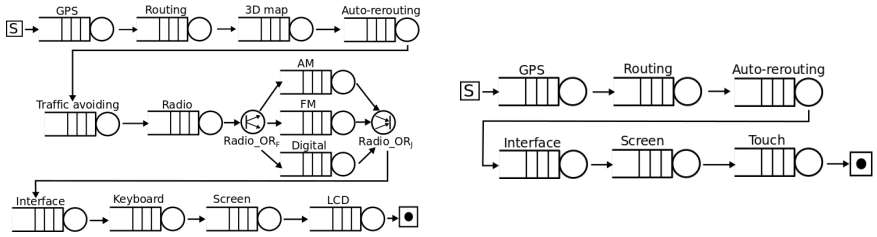
Table 5: Translation from QNSM to QNIM – Slicing method (w is a product and $F_w = features(w)$)

QN element in QNSM	Product w	Sliced QNIM
	$B \in F_w$	
	$B \notin F_w$	
	$B_i \in F_w$	
	$B_{i_1}, \ldots, B_{i_k} \in F_w$	
	$B_{i_1}, \ldots, B_{i_k} \in F_w$	

As an example, we instantiate the QNSM shown in Figure 4 with the two products, i.e., p_{max} that maximizes the reward, and p_{min} that minimizes it, as shown in Table 6. Note that selected features are coupled with a numerical value representing their service time, and it is expressed in units of measurements (e.g., milliseconds or seconds) that are later reflected in the model-based performance results. For example, in Table 6 we can notice that the p_{max} product includes the *3D map* feature whose service time is 86.7 milliseconds. Figure 5 shows the sliced versions of the two QN models representing these specific products. We can observe that the simplified networks are smaller than the non-simplified version; this allows to quickly understand the selected functionalities of the

Table 6: Software products p_{max} and p_{min} produced by FAMA.

Product
p_{max} = {(GPS, 0), (Routing, 0), (3D map, 86.7), (Auto-rerouting, 45.8), (Traffic avoiding, 0), (Radio, 0), (AM, 59.4), (FM, 52.7), (Digital, 54.9), (Interface, 0), (Keyboard, 0), (Screen, 0), (LCD, 0)}
p_{min} = {(GPS, 0), (Routing, 0), (Auto-rerouting, 12.3), (Interface, 0), (Screen, 0), (Touch, 0)}



(a) QNIM-max derived from p_{max}

(b) QNIM-min derived from p_{min}

Figure 5: QNIM-max and QNIM-min (automatically generated from products shown in Table 6).

corresponding system products. Using these two networks, we calculate, by means of JMT [21] – see the box labeled as 5 in Figure 1, the performance results expressing the uncertainty in the attributes' values as lower and upper bounds (i.e., PRLB and PRUB – see the box labeled as 6 in Figure 1) related to the system response time of the selected software products. The obtained performance gap is quite consistent, ranging from 3 to $6.6 \cdot 10^6$ milliseconds. This further motivates the usefulness of model-based performance evaluation of software products, as support for SPL design.

4. Experimentation

This section describes the experimentation conducted to validate our tool-supported framework. The tool (named *FM2QN*), source code, the benchmark

feature models, and the generated queuing networks are publicly available[2]. The approach uses the external tools FAMA[3] and JMT[4]. For the experiments, we used FAMA version 1.1.2 (namely, FAMA Core v1.1.1, FaMa Feature Model v0.9.1, FaMa Attributed Feature Model v1.0.4, and ChocoReasoner v1.1.1) and JMT version 1.0.2. We run all the experiments have on a Linux PC with Intel(R) Xeon CPU 2.3 Ghz and 8 GB of RAM.

4.1. Experimental settings

Our approach takes as input a PAFM (Section 3.1). Although repositories exist for feature models (e.g., the SPLOT repository), to the best of our knowledge no such repository exists for attributed feature models. Therefore, to experiment our approach, we synthesized a set of attributed feature models using the BeTTy tool (version 1.1.1) [30] that allows to randomly generate models with a given number of features, constraints, and some attributes over the features. Specifically, we generated models with the following number of features (NOF): all values between 3 and 10, multiples of 5 between 10 and 50, and multiples of 10 between 50 and 100. For each given NOF, we set the percentage of cross-tree constraints (CTC) – over features – between 5 and 30 (in multiples of 5), and the percentage of extended cross-tree constraints (ECTC) – over attributes – to 5 and 10. The percentage of the other feature model constraints (i.e., optional, mandatory, AND, OR, and ALT) was randomly selected by BeTTy. For each combination of NOF, CTC, and ECTC, we configured BeTTy to annotate features with an attribute *servTime* whose uncertainty is defined over $[L, U]$ intervals, i.e., $L \in \{5, 10, \ldots, 25\}$, $U \in \{10, 15, \ldots, 30\}$, and $L < U$ (to guarantee the correctness of generating minimal and maximal software products). On the basis of these settings, we generated two models for

[2] https://github.com/ERATOMMSD/fm2qn

[3] https://www.isa.us.es/fama/, https://github.com/FaMaFW/FaMA

[4] http://jmt.sourceforge.net

each of the previous combinations, obtaining 6003 feature models[5].

To experiment with our approach on more realistic scenarios, we selected some feature models from the SPLOT repository[6], namely those with more than 100 features. Since SPLOT models do not include attributes, we generated them using again BeTTy. We used the same settings for the attributes and for the constraints over the attributes described above for the synthetic models, obtaining 931 additional feature models, for an overall number of 6934 feature models.

Table 7 reports the characteristics in terms of maximum, minimum, and average number of total features and constraints (i.e., mandatory, optional, AND, etc.) of the benchmark set *BenchSet*. The characteristics of FMs are reported by considering *all* the feature models together, and then aggregating them in five categories depending on the number of features (i.e., 3-24, 25-49, 50-74, 75-100, and 101-176). For each category, Table 7 also reports the number of models with that characteristics, e.g., we evaluate 1080 FMs including a number of features between 75 and 100. All the constraints are reported for the variable number of features, and, as expected, models with more features show also more constraints.

The rationale for grouping the models with more than 100 features is that they all belong to the SPLOT repository, but attribute values are synthetic. The maximum number of features is 176 due to scalability issues when transforming the models from SPLOT to FAMA. In particular, the transformation process requires repeated calls to a constraint solver (i.e., Choco[7]) to remove any inconsistent constraints introduced by BeTTy. For models with more than 176 features (up to 625), this process did not terminate in reasonable time (one day), therefore we decided not to include these models.

Note that, in order to use the generated attributed feature model as PAFM,

[5]Some of the feature models generated by BeTTy were not successfully parsed by FAMA, hence we discarded them.

[6]http://www.splot-research.org

[7]http://www.choco-solver.org

Table 7: Properties of the benchmark set *BenchSet*.

Category (# feat.)		# Feat.	# Mand.	# Opt.	# AND	# OR	# Alt.	# Req.	# Excl.
ALL (6934)	max	176	95	103	42	14	18	44	17
	min	3	0	0	0	0	0	0	0
	avg	53.5	25.6	18.5	9.1	1.82	1.55	3.2	1.51
3-24 (2079)	max	20	16	11	6	5	5	5	5
	min	3	0	0	0	0	0	0	0
	avg	11	4.7	3.2	1.9	0.55	0.42	0.64	0.46
25-49 (1765)	max	45	44	22	12	7	6	10	8
	min	25	2	0	3	0	0	0	0
	avg	35	21	7.8	5.8	1.7	0.64	1.7	1.4
50-74 (1079)	max	70	48	31	16	11	11	14	15
	min	50	14	4	6	0	0	0	0
	avg	60	30	17	9.8	2.7	2.5	2.9	2.6
75-100 (1080)	max	100	68	39	22	14	14	19	17
	min	80	23	11	9	0	0	0	0
	avg	90	45	26	15	4.1	3.7	3.8	3.6
101-176 (931)	max	176	95	103	42	4	18	44	3
	min	103	11	11	6	0	0	1	0
	avg	134	54.6	66.8	24.6	1.15	2.27	11.7	0.41

we need to give a reward semantics to all *OR* and *AND* groups (i.e., parallel and sequential groups, as defined in Defs. 6 and 7). In the following, we consider all groups with either parallel or sequential semantics only. We leave the mixture of these two semantics within one feature model as part of our future work.

To evaluate our approach, we reproduce the process of system modeling and performance evaluation (as shown in Figure 1) for all the benchmark models. To investigate the effect of the two different reward semantics, we apply the process twice: in the first experiment, we consider *OR* and *AND* groups with parallel semantics (i.e., as OR^P and AND^P), and in the second one with sequential semantics (i.e., as OR^S and AND^S). Table 7 shows two instantiations of the benchmark set *BenchSet* with 6934 models each: $BenchSet^P$ with parallel semantics, and $BenchSet^S$ with sequential semantics.

The experiments are conducted as follows. Given a reward semantics (either parallel or sequential), the corresponding benchmark set (i.e., $BenchSet^P$ or $BenchSet^S$) is selected. Then, for each PAFM $\overline{fm}$:

- we generate the QNSM (a queueing network qn and a set of constraints $Constr$) using the approach presented in Section 3.2;
- we then generate the products p_{max} and p_{min} having the *maximal* and *minimal* reward (see Def. 5) for *servTime* using FAMA. Note that FAMA implicitly uses a sequential reward semantics for maximization and minimization (as it sums the rewards of the features' attributes). Therefore, to obtain from FAMA products with maximal and minimal rewards in FMs with parallel reward semantics, we add an attribute *servTimePar* to the model. The value of *servTimePar* for a feature is given as the maximum among the values of *servTimePar* of its children (see Def. 5) for non-leaf nodes, and it is equal to *servTime* in leaf nodes. This way, our parallel semantics is correctly interpreted by FAMA without modifying its internal modules;
- we then instantiate qn with the two products p_{max} and p_{min} as described in Section 3.3. Specifically, we use the *setting method* described in Section 3.3.1 that includes the settings of feature costs and routing probabilities, obtaining QNIM-max and QNIM-min;
- finally, we simulate these two queueing network models with JMT to evaluate the average system response time. At this stage, we are not considering further performance indices (e.g., resource utilization, service throughput, etc.) for the sake of illustration, but all the generated QN models can be used to get further performance indicators.

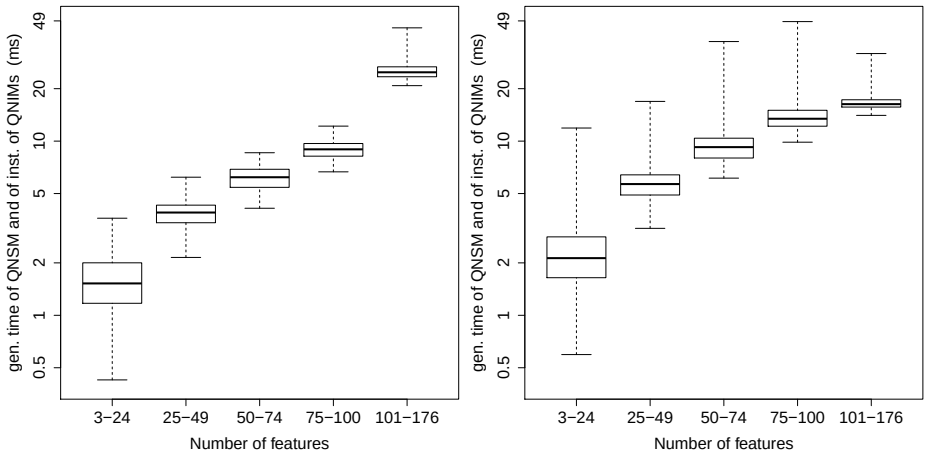
4.2. Quantitative evaluation

Table 8 reports the efficiency of the proposed approach while considering the parallel and the sequential semantics separately. Specifically, it reports the

Table 8: Experimental Results: efficiency of the proposed approach.

		QNSM gen. & QNIM inst.	Selection		Simulation	
			p_{max}	p_{min}	QNIM-max	QNIM-min
Par.	Tot (mins)	1	12149.8	4692.6	16980.8	9580.7
sem.	Avg (secs)	0.01	105.2	40.6	147	82.9
Seq.	Tot (mins)	0.9	14199.8	5415.4	11406.8	6138.4
sem.	Avg (secs)	0.01	122.9	46.9	98.7	53.1

total and average generation time of a QNSM including the instantiation time of QNIM-max and QNIM-min. This timing information is provided for both the p_{max} and p_{min} software products. We also report the total and average simulation time of the QNIM-max and QNIM-min models (dispersion of the results will be shown later in Figures 6, 7, and 8). The results shown in Table 8 indicate that the time needed by the proposed tool-based framework to transform a PAFM into a QNSM (and to instantiate it into a QNIM) is negligible w.r.t. the time taken for generating products from PAFM and for simulating QNIM. For example, for the whole set of 6934 feature models (where all groups are set with a parallel semantics), the overall time to generate the QNSM is one minute. The selection of software products takes a bit longer, averaging at 105.2 seconds for p_{max} and 40.6 seconds for p_{min}, respectively. This difference may be due to FAMA that, when selecting the minimal product, performs some internal optimization on the considered features before calling the external constraint programming solver which is expensive. Simulation of these models, instead, goes from 147 to 82.9 seconds for QNIM-max and QNIM-min, respectively. The sequential semantics results to be more efficient in the simulation of models, but quite similar (slightly slower) when considering their generation. Note that our approach does not focus on the optimization. Optimization is performed by FAMA, and so we inherit its scalability issues [5]. Moreover, FAMA internally exploits Choco as constraint solver, hence our approach guarantees the correctness of the generated products and the fulfillment of the given



(a) Sequential semantics

(b) Parallel semantics

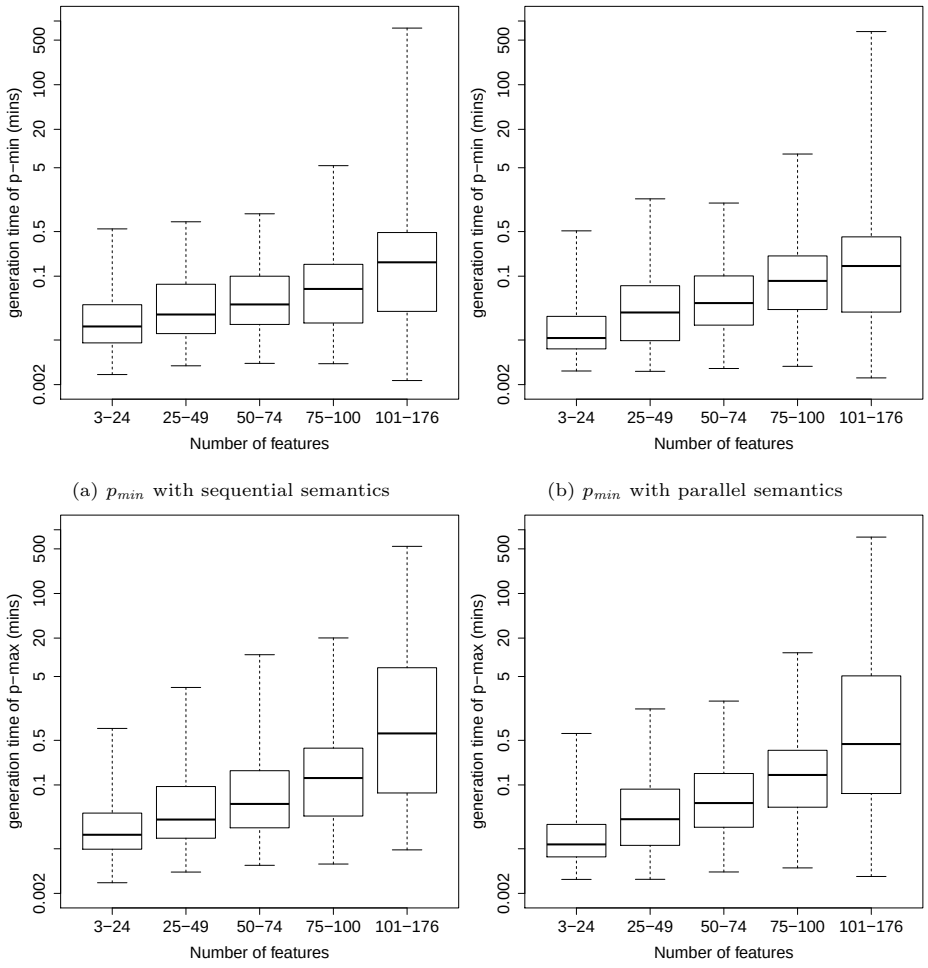
Figure 6: Generation time of QNSM and instantiation of QNIM.

constraints. In contrast, other methodologies make use of multi-objective evolutionary algorithms [31, 32, 33] to efficiently address the SPL configuration optimization problem. However, these approaches produce solutions that may violate predefined constraints.

In the following, we provide a more detailed evaluation by assessing the influence of the model size on the proposed process. We first measure the time required by our tool for generating a QNSM from a PAFM and then instantiating it in QNIMs. Figure 6 shows the distribution of the generation time (in milliseconds) for each group of models, for both the sequential and the parallel semantics. To smooth out potential sources of bias, we repeated the generation of QNSM one hundred times, reporting the average results per model. We can observe that, as expected, the generation time grows with the model size for both semantics. Parallel semantics is sometimes slightly slower as it usually generates more complex networks. However, the overall process is quite fast as it requires at most, for the biggest model, less than 50 milliseconds.

To evaluate the product generation time required by FAMA[8], Figure 7 shows

[8]Due to the high latencies, we only perform one run for the experiments related to the



(a) p_{min} with sequential semantics

(b) p_{min} with parallel semantics

(c) p_{max} with sequential semantics

(d) p_{max} with parallel semantics

Figure 7: Product generation time with FAMA [5].

the distribution of the generation time (in minutes) for maximal and minimal products p_{max} and p_{min}. Figures 7a and 7c show the values for the models with sequential semantics, whereas Figures 7b and 7d for models with parallel seman-

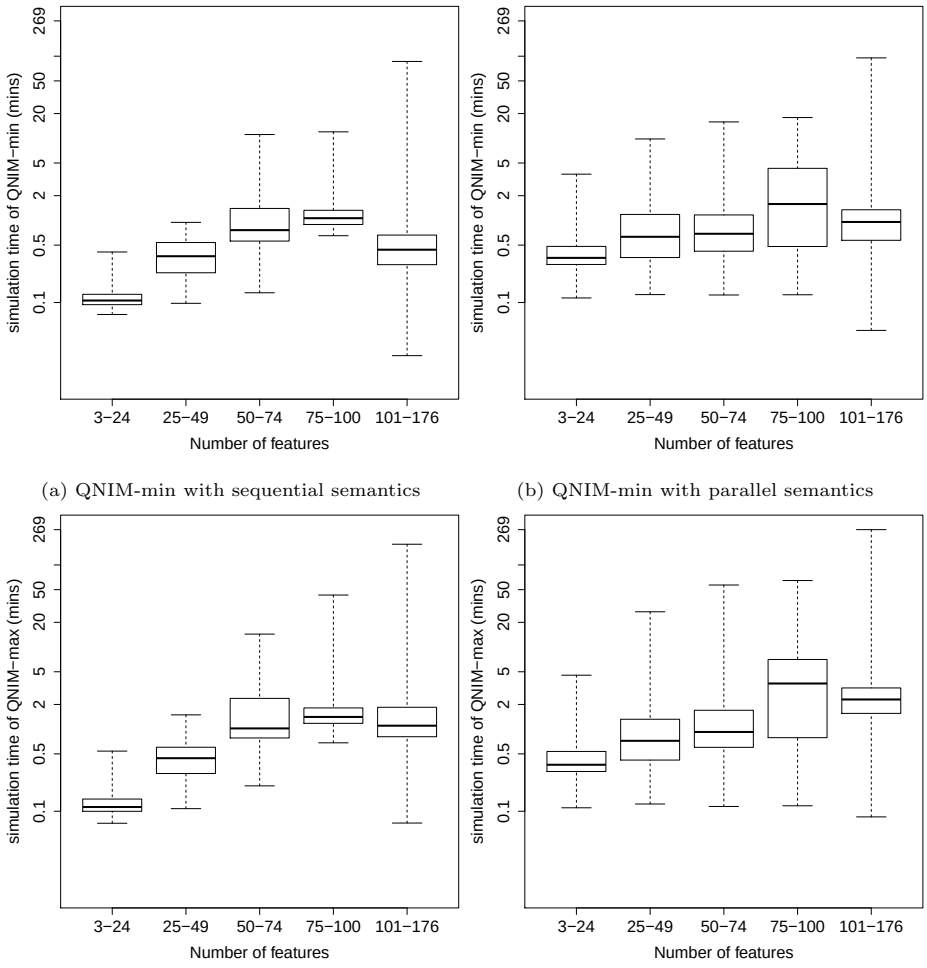
products generation and the simulation of the generated queueing networks. This is discussed as threat of validity (see Section 5) and motivated by the fact that we are not interested in assessing the performance of FAMA and JMT, that are external tools, but only to what extent they affect the whole process of model-based performance analysis.

tics. Similarly to the previous experiment, we can observe that, as expected, the generation time grows with the size of the model. We can notice slight differences when comparing minimal and maximal products. We do not observe, instead, a significant difference between the generation with parallel and sequential semantics. The overall process of product generation requires at most 20 minutes for models up to 100 features, whereas, as expected, takes longer (up to 996 minutes) when considering larger models, i.e., up to 176 features.

The generation of minimal and maximal products by FAMA relies on constraint satisfaction (delegated to the built-in Choco solver). As stated in Section 1, it is worth to remark that software products can be either provided by the user or obtained by applying some heuristics, without being supported by third-party tools. Our approach accepts any valid product that can be generated by alternative methodologies (e.g., selected by the user through a feature model configurator [28]). This implies that the time for generating software products strongly depends on the adopted strategy. Without third-party tools, this time may be even neglected in the evaluation of the overall process when products are selected manually.

Figure 8 shows the simulation time of JMT over the generated QNIM-max and QNIM-min models. Figures 8a and 8c show the simulation time for the models with sequential semantics, whereas Figures 8b and 8d for those models with parallel semantics. Similarly to the previous experiments, for both semantics the simulation time grows with the size of the model. We can observe different distributions for the two semantics. For the sequential case, the simulation is on average slightly faster; this is due to the fact that the queueing networks produced for modeling the sequential semantics do not contain forks and joins, hence they are easier to solve [34]. Models with parallel semantics show a higher variability, leading to larger regions in the boxplots shown in Figures 8b and 8d. The overall process may turn out to be slow, from 50 minutes (when considering up to 100 features) and up to 269 minutes (when looking at the larger models with up to 176 features).

Figure 9 shows the system response time (RT). Figure 9a and Figure 9c



(a) QNIM-min with sequential semantics

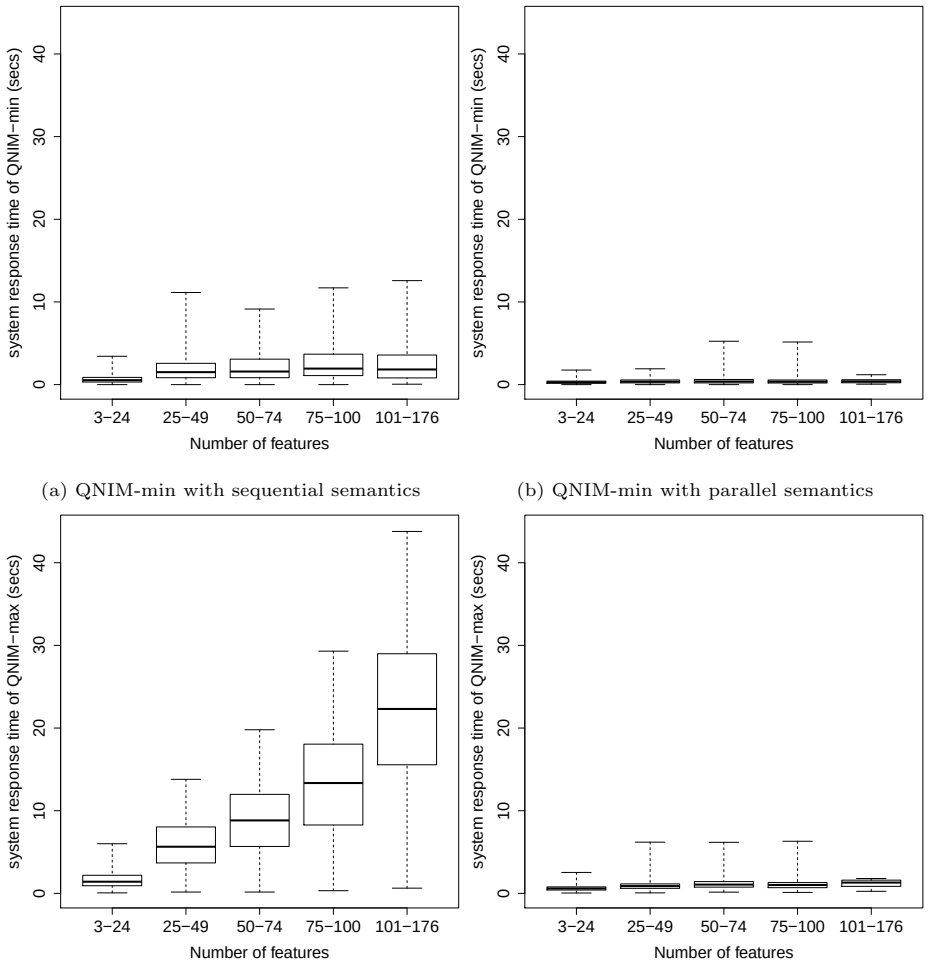
(b) QNIM-min with parallel semantics

(c) QNIM-max with sequential semantics

(d) QNIM-max with parallel semantics

Figure 8: Simulation time with JMT [21].

depict the sequential semantics for QNIM-min and QNIM-max, respectively. Figure 9b and Figure 9d report their parallel counterpart. As expected, systems with sequential semantics have a higher RT as the execution of activities is not parallelized. This is confirmed by the experimental results: the maximum RT for minimal and maximal products are 12.6 and 43.8 seconds for the sequential semantics (Figures 9a and 9c), while they are 5.25 and 6.31 seconds for the parallel semantics (Figures 9b and 9d). We do not observe significant variations



(a) QNIM-min with sequential semantics

(b) QNIM-min with parallel semantics

(c) QNIM-max with sequential semantics

(d) QNIM-max with parallel semantics

Figure 9: System response time (RT).

for the parallel semantics, as all the system response times vary in tiny intervals. On the contrary, we can observe that QNIM-min models with sequential semantics produce values varying up to 12.6 seconds, even if the median and average values (across all the models) are 1.14 and 1.75 seconds, respectively. This trend is even more evident while considering QNIM-max models with sequential semantics, with response time up to 43.8 seconds, and median and average values (across all the models) of 5.88 and 8.63 seconds, respectively.

In this paper, we consider the selection and simulation of minimal and maximal products as an example of model-based performance analysis that can be done starting from the QNSM. We are also interested in assessing whether the products that FAMA identifies as maximal are significantly different (from a performance perspective) than those identified as minimal products. To this aim, we apply the Mann-Whitney U Test to assess the significance of results (i.e., the difference between the distributions of minimal and maximal products), and the A12 statistics to assess the strength of such significance, as explained in [35]. The tests are applied considering the system response time of minimal and maximal products (with sequential semantics), and assess that the maximal products are significantly larger than the minimal ones (p-value=0 and A-12=0.83). We obtained the same results for the maximal and minimal products under parallel semantics (p-value=0 and A-12=0.85). These results confirm that the minimal and maximal products are indeed able to show a large difference in the system performance.

Our experimentation demonstrates how the model-based performance analysis can support software engineers in the process of understanding the different performance characteristics among minimal and maximal products.

5. Threats to validity

Besides inheriting all limitations of the underlying software product lines and software performance engineering research areas [24, 22, 36, 37], our approach exhibits the following threats to validity [38].

Threats to construct validity. Our approach primarily aims at providing an automated technique for the generation of performance models. As such, correctness of the queuing network models that we generate represents a threat to construct validity. As a first syntactic check we made sure that every model could be successfully parsed by the tool used for the analysis (i.e., JMT). However, the networks generated by our approach are directed acyclic graphs by construction, while JMT does not require every node of the network to be

reachable from the root. We thus verified such connectivity separately. We also checked the correct instantiation of a QNIM from a QNSM following the setting and slicing methods (Section 3.3), to respectively enforce that only the nodes representing product features have an incoming probability of one, and that the network exclusively contains nodes related to the product's features.

One aspect we meant to evaluate with our experiments is the *efficiency* of the proposed technique measured in terms of computational effort. However, simply measuring the overall time might not have been indicative enough in our case, as it depends on three different components, i.e., the generation time for different feature models of increasing complexity, the product generation time taken by FAMA, and the simulation time with JMT. We thus measured these separately and also varied the complexity of the input, to evaluate the impact of complexity on each measure.

Threats to internal validity. Our approach relies on external components for product generation and simulation, and is thus exposed to instrumentation threats related to the execution of the experiments. In that respect, we would like to observe that both FAMA and JMT are well-known tools that have been around for a long time now, and are still regularly maintained.

We measured the elapsed time of every component in our workflow with standard Linux tools. These tools are known to introduce measurement error, for instance when the machine performs at the same time other computations that interfere with the measurements. To avoid that, we made sure that the machine was otherwise idle.

Threats to external validity. We are aware that the findings from our experiments may not immediately transfer to different domains, and thus other families of software product lines. To mitigate this threat, we used generated synthetic feature models considering different system properties (e.g., the number of features, the number of mandatory vs optional features, etc.) to obtain a variety of models that may be representative for different scenarios. Such synthetic models are useful to experiment the approach under several different conditions,

but may still not be representative of real-world feature models. For this reason, in the experiments we additionally included realistic feature models from the SPLOT repository.

Threats to conclusion validity. These threats concern issues that may affect the ability to draw the correct conclusion about relations between the settings and the outcome [38]. To ensure the reliability of the approach and the validity of data, we made publicly available the source code of our tool and the adopted benchmarks (see https://github.com/ERATOMMSD/fm2qn). Moreover, since the approach relies on external tools for product generation (FAMA) and *qn* simulation (JMT), we also report the exact versions of the tools used in our experiments (see Section 4), to guarantee the full reproducibility of the experimental results.

6. Related work

Feature models [4] allow to describe families of products called Software Product Lines (SPLs) using a tree-like structure, and they have been originally conceived to focus on variability modeling [39, 40]. Different tools have been developed for this scope, e.g., FeatureIDE [26] and FAMILIAR [41]. More recently, some effort has been devoted to automatically or semi-automatically map feature models into abstractions with execution semantics such as Business Process Execution Language (BPEL) [42, 43]. Some approaches have also incorporated variability into Business Process Model and Notation (BPMN) to form variable business processes and template-based business process families [44, 45]. This supports our effort of manipulating feature models and extending their semantics to enable a quantitative performance evaluation, thus to get a glimpse on the performance characteristics of SPLs.

In the sequel of the section we discuss the state of the art in the literature dealing with the optimization of feature models. Although this paper focuses on performance concerns, other non-functional (NF) properties, are also reviewed to provide a wider overview of current research trends.

Strategies to efficiently analyze the performance of software variants have been recently surveyed in [29]. The developed techniques are: sampling the variant space [46]; generation of test suites covering all variants [47]; predicting the performance of variants analytically [48]. Our work is closely related to [48] where Coxian distributions are considered and the family-based analysis is efficiently performed by solving ordinary differential equations (ODE). We are also interested in providing analytical performance predictions, but differently from [48] our focus is in transforming feature models (with performance-related uncertainties, e.g., the service time) in performance analytical models (i.e., Fork-join QNs), the efficiency of the analysis is not tackled.

Essential concepts on software families and software product lines in industrial practice are reviewed in [49], where the problem of pointing out dependencies between features and inform developers about them is raised. In [50] a literature review is conducted to investigate how the quality attribute variability is considered in software product lines, and it turns out that different approaches suit specific quality attributes differently, empirical evidence in industrial contexts is lacking. In [27] an automated reasoning on feature models that takes into account functional and extra-functional features is proposed by using constraint programming; however, it does not take into account the ordering among software products. Existing approaches for specifying variation in quality attributes are surveyed in [51], where some requirements are listed, e.g., automatic reasoning, optionality, qualitative or quantitative analysis, etc. Interestingly, optionality at product line level (i.e., in a product one quality attribute may be important and in another this attribute may not be required) results to be almost neglected. In [52] the problem of selecting features to achieve customer requirements is formalised using 0-1 programming in order to efficiently provide a solution, but the interaction of features is not considered and its optimality for quality attributes is doubtful. In [53] an approximation algorithm for selecting a set of architectural features adhering to resource constraints is proposed, but it does not deal with quality attributes derived from the selection of features and its optimality is not guaranteed. In [54] an artificial intelligence

technique is presented to automatically select suitable features that satisfy both the stakeholders functional and non-functional preferences; however, the non-functional annotations are arbitrarily defined and not validated on the basis of selected features. In [55] various search-based software engineering methods are adopted to optimize the values of user preferences in software product lines; however, these algorithms provide an approximate solution that is affected by the parameters of crossover and mutation operators. In [56] the problem of optimizing multiple objectives in large software product lines is tackled, but the search process is affected by constraint solving and genetic searching that may result inefficient while dealing with quality attributes.

Table 9 schematically reports the related works dealing with performance, and other non-functional properties. First column shows the considered property, second column lists the different approaches, third and fourth columns report pros and cons of these approaches, respectively. This literature review includes all the papers we found more relevant when compared to our approach, and it is far from being exhaustive.

In [57] a variability-aware approach to performance prediction for configurable software systems is presented. It builds upon random samples and makes use of statistical learning techniques to build a performance model that represents the correlation between feature selections and performance. In [58] an approach for deriving performance-influence models for configurable systems is proposed. Machine-learning techniques are combined with sampling heuristics for binary and numeric configuration options for improving the accuracy of the models. In [59] sampling strategies for performance prediction of configurable systems are adopted, and the heuristic is based on feature frequencies to guide the initial sample generation of projective sampling. Being a learning technique, similarly to [58], it is necessary to find a balance between measurement effort and prediction accuracy. All these three approaches [57, 58, 59] require performance measurements to derive models and adopt learning procedures to derive performance models. On the contrary, our approach does not require performance knowledge, it automatically builds a performance model from design

Table 9: Overview of related works (NF: non-functional property. App: approach).

NF	*App*	*Pros*	*Cons*
Performance	[57]	performance modeling based on statistical learning	94% of accuracy measuring randomly selected configurations
	[58]	performance-influence model is derived for configurable systems	tradeoff between measurement effort and prediction accuracy
	[59]	sampling strategies and heuristics, e.g., features frequencies	tradeoff between measurement effort and prediction accuracy
	[60]	model-based performance analysis from feature models	manually built queueing network performance models
	[61]	performance ad-hoc annotations in feature models	LQN performance models are adopted for analysis
	[62]	model-based performance analysis for operating systems	limited to model variabilities of hardware features
	[63]	ODE-based performance analysis and fluid limits	variations expressed at the level of the performance model
Reliability	[64]	feature-family-based strategy for efficient reliability analysis	performance speedups do not consider some system characteristics
	[65]	probabilistic model checking for reliability evaluations	scalability issues may arise when increasing system size
	[66]	feature models are used to derive reliability properties	user preferences are not considered in the optimization
Security	[67]	security requirements engineering process for software product lines	security checks on system products conform to standards
	[3]	security annotations are added in feature models	scalability issues while solving the multi-objective problem

specifications, and performance prediction results are derived afterwards.

In [62] a performance model is proposed for general-purpose operating system schedulers while considering Symmetric Multi Processing (SMP) environments. Our approach differs from this since it also includes software variabilities. In our previous work, in [68] we investigate the influence of uncertain parameters on system performance and in [60] software (e.g., services) and hardware

(e.g., single vs multi-core processors) variable features are considered. In both these two approaches [68, 60] the performance models are manually built and no performance-based ordering is provided as support to system stakeholders.

In [63, 69] the idea of SPL is exploited to perform family-based performance analysis while leveraging the commonalities across variants. A specific notation is introduced to annotate parametric and structural changes that are later translated in ordinary differential equations (ODE), and the performance analysis is based on the theory of fluid limits [70]. Variations are expressed at the level of the performance model since the goal of [63, 69] is to demonstrate the efficiency of the analysis. On the contrary, our approach introduces annotations in feature models to correlate alternative system designs with their performance characteristics.

To automatically derive performance models we adopted the methodology presented in [61], where the addition of performance annotations on the feature model contribute to the specification of performance models representing specific software products. However, in [61] the transformation is not automated but specified towards Layered Queueing Networks (LQN), whereas in this paper we focus on automatically transforming feature models into Queueing Network (QN) performance models.

In [64] a feature-family-based strategy for reliability analysis of product lines is proposed, and an empirical study demonstrates the efficiency of the analysis vs other state-of-the-art methodologies. As part of future work, authors discuss the investigation of the performance impact due to some system characteristics (e.g., number of decision nodes). In [65] probabilistic model checking techniques are used to verify reliability properties of different configurations of a software product line. This way, software engineers are supported in the task of evaluating the non-functional characteristics of design solutions in the early stages of development. In [66] feature models are used to formulate the redundancy allocation problem and solutions consider varying trade-offs between cost and reliability. However, partial knowledge about possible user preferences is not integrated in the formulation of the optimization problem.

In [67] a security standard-based process that deals with security requirements from the early stages of SPL development is proposed. It is based on security requirements techniques, such as UMLSec that basically consists in annotating UML models with security-related information in order to unambiguously define the properties to achieve, e.g., data confidentiality in a network link. However, the security analysis only verifies if system products conform to standards. In [3] security information is annotated in feature models and considered for multi-objective optimization process, where other non-functional attributes are also considered. The goal is to automatically determine the selection of features to optimize desired quality attributes of the resulting product; however, there are scalability limitations for the current optimization infrastructure.

7. Conclusion

This paper presented an automated framework that allows to transform feature models into queuing networks and enables the performance evaluation of SPLs. This way, software engineers are supported in identifying sources of performance issues and guided in the process of selecting features (along with their intrinsic uncertainty) that do not violate performance requirements. The approach has been validated through a set of 6934 feature models, and we found that the timing of generating queueing networks is negligible with respect to their simulation needed to get performance indicators. Experimental results are promising and encourage future work in this direction.

First, currently in the tool the semantics for all the *OR* and *AND* groups in a feature model is either set to sequential or parallel, but their mixture is not enabled. The user may need to specify different semantics within the feature model, and as future work we plan to allow the specification of the desired semantics for each group. This implies to provide a domain specific language (DSL) able to express feature models with mixed semantics, and it is also necessary to develop a translator to FAMA models in order to generate minimal and maximal products. Second, we plan to integrate FM2QN with other related

techniques, such as feature-interaction detection [71] and measurements-based approaches [72], to reduce the model-based performance analysis effort in wider application domains. For example, by combining various feature optimization heuristics [73] and integrating knowledge of feature interactions [74], the space of valid products is reduced and our model-based performance analysis of software product lines may result to be optimized. Further experimentation is needed to investigate the benefits of this integration. Third, we plan to extend the performance evaluation to further metrics (e.g., resources utilization, throughput of services, etc.) and conduct a trade-off analysis among them (possibly with ensemble methods already proposed in the literature, e.g., [75, 76]), thus to better investigate the system performance of the generated products.

Finally, our approach is focused on the performance evaluation, but it may be interesting to study further extra-functional characteristics of software systems, such as reliability and security. This implies to transform feature models into further quality-based models, such as fault trees for reliability, and to conduct a trade-off analysis among multiple quality attributes.

Acknowledgments

P. Arcaini is supported by ERATO HASUO Metamathematics for Systems Design Project (No. JPMJER1603), JST. Funding Reference number: 10.13039/501100009024 ERATO. This work has been partially funded by MIUR projects PRIN 2017FTXR7S IT-MATTERS (Methods and Tools for Trustworthy Smart Systems) and PRIN 2017TWRCNB SEDUCE (Designing Spatially Distributed Cyber-Physical Systems under Uncertainty).

References

[1] C. Lengauer, S. Apel, Feature-oriented system design and engineering, Int. J. Software and Informatics 5 (1-2) (2011) 231–244.

[2] S. Apel, C. Kästner, An overview of feature-oriented software development, Journal of Object Technology 8 (5) (2009) 49–84.

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

[4] D. Batory, Feature models, grammars, and propositional formulas, in: Proceedings of the International Software Product Line Conference (SPLC), Vol. 3714, 2005, pp. 7–20.

[5] P. Trinidad, D. Benavides, A. Ruiz-Cortés, S. Segura, A. Jimenez, FAMA framework, in: Proceedings of the International Software Product Line Conference (SPLC), 2008, pp. 359–359.

[6] L. M. S. Tran, F. Massacci, An approach for decision support on the uncertainty in feature model evolution, in: Proceedings of the International Conference on Requirements Engineering (RE), 2014, pp. 93–102.

[7] M. H. ter Beek, A. Legay, A. L. Lafuente, A. Vandin, Statistical analysis of probabilistic models of software product lines with quantitative constraints, in: Proceedings of the International Conference on Software Product Line (SPLC), 2015, pp. 11–15.

[8] M. H. ter Beek, A. Legay, Quantitative variability modeling and analysis, in: Proceedings of the International Workshop on Variability Modelling of Software-Intensive Systems (VAMOS), 2019, pp. 13:1–13:2.

[9] M. Harman, P. O'Hearn, From start-ups to scale-ups: opportunities and open problems for static and dynamic program analysis, in: Proceedings of the International Working Conference on Source Code Analysis and Manipulation (SCAM), 2018, pp. 1–23.

[10] P. Clements, L. Bass, R. Kazman, G. Abowd, Predicting software quality by architecture-level evaluation, in: Proceedings of the International Con-

ference on Software Quality, Reliability, and Security (QRS), Vol. 5, 1995, pp. 485–497.

[11] C. Trubiani, A. Ghabi, A. Egyed, Exploiting traceability uncertainty between software architectural models and performance analysis results, in: Proceedings of the European Conference on Software Architecture (ECSA), 2015, pp. 305–321.

[12] R. Tawhid, D. C. Petriu, Automatic derivation of a product performance model from a software product line model, in: Proceedings of the International Conference on Software Product Line (SPLC), 2011, pp. 80–89.

[13] S. Bernardi, J. Merseguer, D. C. Petriu, A dependability profile within marte, Software & Systems Modeling 10 (3) (2011) 313–336.

[14] V. Cortellessa, R. Mirandola, PRIMA-UML: a performance validation incremental methodology on early UML diagrams, Science of Computer Programming 44 (1) (2002) 101–129.

[15] S. Balsamo, M. Marzolla, Performance evaluation of UML software architectures with multiclass queueing network models, in: Proceedings of the International Workshop on Software and Performance (WOSP), 2005, pp. 37–42.

[16] C. U. Smith, C. M. Lladó, R. Puigjaner, Performance model interchange format (PMIF 2): A comprehensive approach to queueing network model interoperability, Perform. Eval. 67 (7) (2010) 548–568.

[17] V. Cortellessa, A. D. Marco, P. Inverardi, Model-Based Software Performance Analysis, Springer, 2011.

[18] E. D. Lazowska, J. Zahorjan, G. Scott Graham, K. C. Sevcik, Computer System Analysis Using Queueing Network Models, Prentice-Hall, Inc., Englewood Cliffs, 1984.

[19] L. Kleinrock, Theory, Volume 1, Queueing Systems, Wiley-Interscience, New York, NY, USA, 1975.

[20] P. Arcaini, C. Trubiani, Collaborative development of feature models and evaluation of performance bounds, in: Proceedings of the ACM Symposium on Applied Computing (SAC), 2017, pp. 1162–1167.

[21] M. Bertoli, G. Casale, G. Serazzi, JMT: Performance engineering tools for system modeling, SIGMETRICS Perform. Eval. Rev. 36 (4) (2009) 10–15.

[22] T. Thüm, S. Apel, C. Kästner, I. Schaefer, G. Saake, A classification and survey of analysis strategies for software product lines, ACM Computing Surveys (CSUR) 47 (1) (2014) 6.

[23] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, A. S. Peterson, Feature-oriented domain analysis (foda) feasibility study, Tech. rep., Carnegie-Mellon Univ. Pittsburgh Software Engineering Institute (1990).

[24] D. Batory, D. Benavides, A. Ruiz-Cortes, Automated analysis of feature models: challenges ahead, Communications of the ACM 49 (12) (2006) 45–47.

[25] D. Benavides, S. Segura, A. Ruiz-Cortés, Automated analysis of feature models 20 years later: A literature review, Information Systems 35 (6) (2010) 615–636.

[26] T. Thüm, C. Kästner, F. Benduhn, J. Meinicke, G. Saake, T. Leich, FeatureIDE: An extensible framework for feature-oriented software development, Science of Computer Programming 79 (0) (2014) 70–85.

[27] D. Benavides, P. Trinidad, A. Ruiz-Cortés, Automated reasoning on feature models, in: Proceedings of the International Conference on Advanced Information Systems Engineering (CAiSE), 2005, pp. 491–503.

[28] J. Meinicke, T. Thüm, R. Schröter, F. Benduhn, T. Leich, G. Saake, Mastering Software Variability with FeatureIDE, Springer, 2017.

[29] T. Thüm, A. van Hoorn, S. Apel, J. Bürdek, S. Getir, R. Heinrich, R. Jung, M. Kowal, M. Lochau, I. Schaefer, J. Walter, Performance analysis strategies for software variants and versions, in: Managed Software Evolution, Springer, 2019, pp. 175–206.

[30] S. Segura, J. A. Galindo, D. Benavides, J. A. Parejo, A. Ruiz-Cortés, BeTTy: Benchmarking and Testing on the Automated Analysis of Feature Models, in: Proceedings of the International Workshop on Variability Modeling of Software-Intensive Systems (VaMoS), 2012, pp. 63–71.

[31] Y. Xiang, Y. Zhou, Z. Zheng, M. Li, Configuring software product lines by combining many-objective optimization and SAT solvers, ACM Transactions on Software Engineering and Methodology (TOSEM) 26 (4) (2018) 14.

[32] X. Lian, L. Zhang, J. Jiang, W. Goss, An approach for optimized feature selection in large-scale software product lines, Journal of Systems and Software 137 (2018) 636–651.

[33] J. Guo, K. Shi, To preserve or not to preserve invalid solutions in search-based software engineering: a case study in software product lines, in: Proceedings of the 40th International Conference on Software Engineering, 2018, pp. 1027–1038.

[34] A. Thomasian, Analysis of fork/join and related queueing systems, ACM Computing Surveys 47 (2) (2015).

[35] A. Arcuri, L. C. Briand, A practical guide for using statistical tests to assess randomized algorithms in software engineering, in: Proceedings of the International Conference on Software Engineering (ICSE), 2011, pp. 1–10.

[36] T. Hall, S. Beecham, D. Bowes, D. Gray, S. Counsell, A systematic literature review on fault prediction performance in software engineering, IEEE Transactions on Software Engineering 38 (6) (2012) 1276–1304.

[37] M. Woodside, G. Franks, D. C. Petriu, The future of software performance engineering, in: Proceedings of the International Workshop on Future of Software Engineering (FSE), 2007, pp. 171–187.

[38] C. Wohlin, P. Runeson, M. Hst, M. C. Ohlsson, B. Regnell, A. Wessln, Experimentation in Software Engineering, Springer Publishing Company, Incorporated, 2012.

[39] L. Chen, M. Ali Babar, N. Ali, Variability management in software product lines: a systematic review, in: Proceedings of the International Conference on Software Product Line (SPLC), 2009, pp. 81–90.

[40] I. Schaefer, Variability modelling for model-driven development of software product lines, in: Proceedings of the International Workshop on Variability Modelling of Software-Intensive Systems (VAMOS), 2010, pp. 85–92.

[41] M. Acher, P. Collet, P. Lahire, R. B. France, FAMILIAR: A domain-specific language for large scale management of feature models, Science of Computer Programming 78 (6) (2013) 657–681.

[42] T. Nguyen, A. Colman, A feature-oriented approach for web service customization, in: Proceedings of the International Conference on Web Services (ICWS), 2010, pp. 393–400.

[43] Z. Peng, J. Wang, K. He, M. Tang, Web service customization based on service feature model, in: Proceedings of the International Conference on Services Computing (ICSC), 2015, pp. 632–639.

[44] A. Schnieders, F. Puhlmann, Variability mechanisms in e-business process families., BIS 85 (2006) 583–601.

[45] A. Schnieders, F. Puhlmann, Variability modeling and product derivation in e-business process families, in: Technologies for business information systems, Springer, 2007, pp. 63–74.

[46] M. Varshosaz, M. Al-Hajjaji, T. Thüm, T. Runge, M. R. Mousavi, I. Schaefer, A classification of product sampling for software product lines, in: Proceedings of the International Conference on Systems and Software Product Line (SPLC), 2018, pp. 1–13.

[47] J. Bürdek, M. Lochau, S. Bauregger, A. Holzer, A. Von Rhein, S. Apel, D. Beyer, Facilitating reuse in multi-goal test-suite generation for software product lines, in: Proceedings of the International Conference on Fundamental Approaches to Software Engineering (FASE), 2015, pp. 84–99.

[48] M. Kowal, M. Tschaikowski, M. Tribastone, I. Schaefer, Scaling size and parameter spaces in variability-aware software performance models, in: Software Engineering, 2016, pp. 33–34.

[49] M. Ribeiro, P. Borba, C. Brabrand, Software Families, Software Products Lines, and Dataflow Analyses, Emergent Interfaces for Feature Modularization, Springer, 2014, pp. 7–21.

[50] V. Myllärniemi, M. Raatikainen, T. Männistö, A systematically conducted literature review: Quality attribute variability in software product lines, in: Proceedings of the International Software Product Line Conference (SPLC), 2012, pp. 41–45.

[51] L. Etxeberria, G. S. Mendieta, L. Belategi, Modelling variation in quality attributes, in: Proceedings of the International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS), 2007, pp. 51–59.

[52] J. Li, X. Liu, Y. Wang, J. Guo, Formalizing feature selection problem in software product lines using 0-1 programming, in: Practical Applications of Intelligent Systems, Springer, 2011, pp. 459–465.

[53] J. White, B. Dougherty, D. C. Schmidt, Selecting highly optimal architectural feature sets with filtered cartesian flattening, Journal of Systems and Software 82 (8) (2009) 1268–1284.

[54] S. Soltani, M. Asadi, D. Gašević, M. Hatala, E. Bagheri, Automated planning for feature model configuration based on functional and non-functional requirements, in: Proceedings of the International Software Product Line Conference (SPLC), 2012, pp. 56–65.

[55] A. S. Sayyad, T. Menzies, H. Ammar, On the value of user preferences in search-based software engineering: a case study in software product lines, in: Proceedings of the International Conference on Software Engineering (ICSE), 2013, pp. 492–501.

[56] C. Henard, M. Papadakis, M. Harman, Y. Le Traon, Combining multi-objective search and constraint solving for configuring large software product lines, in: Proceedings of the International Conference on Software Engineering (ICSE), 2015, pp. 517–528.

[57] J. Guo, K. Czarnecki, S. Apel, N. Siegmundy, A. Wasowski, Variability-aware performance prediction: A statistical learning approach, in: Proceedings of the International Conference on Automated Software Engineering (ASE), 2013, pp. 301–311.

[58] N. Siegmund, A. Grebhahn, S. Apel, C. Kästner, Performance-influence models for highly configurable systems, in: Proceedings of the Joint Meeting on Foundations of Software Engineering (FSE), 2015, pp. 284–294.

[59] A. Sarkar, J. Guo, N. Siegmund, S. Apel, K. Czarnecki, Cost-efficient sampling for performance prediction of configurable systems (t), in: Proceedings of the International Conference on Automated Software Engineering (ASE), 2015, pp. 342–352.

[60] L. Etxeberria, C. Trubiani, V. Cortellessa, G. Sagardui, Performance-based selection of software and hardware features under parameter uncertainty, in: Proceedings of the International Conference on Quality of Software Architectures (QoSA), 2014, pp. 23–32.

[61] R. Tawhid, D. Petriu, User-friendly approach for handling performance parameters during predictive software performance engineering, in: Proceedings of the International Conference on Performance Engineering (ICPE), 2012, pp. 109–120.

[62] J. Happe, H. Groenda, M. Hauck, R. H. Reussner, A prediction model for software performance in symmetric multiprocessing environments, in: Proceedings of the International Conference on the Quantitative Evaluation of Systems (QEST), 2010, pp. 59–68.

[63] M. Kowal, M. Tschaikowski, M. Tribastone, I. Schaefer, Scaling size and parameter spaces in variability-aware software performance models, in: Proceedings of the International Conference on Automated Software Engineering (ASE), 2015, pp. 407–417.

[64] A. Lanna, T. Castro, V. Alves, G. Rodrigues, P.-Y. Schobbens, S. Apel, Feature-family-based reliability analysis of software product lines, Information and Software Technology 94 (2018) 59–81.

[65] C. Ghezzi, A. M. Sharifloo, Verifying non-functional properties of software product lines: Towards an efficient approach using parametric model checking, in: Proceedings of the International Conference on Software Product Line (SPLC), 2011, pp. 170–174.

[66] P. Limbourg, H.-D. Kochs, Multi-objective optimization of generalized reliability design problems using feature models – A concept for early design stages, Reliability Engineering & System Safety 93 (6) (2008) 815–828.

[67] D. Mellado, E. Fernández-Medina, M. Piattini, Towards security requirements management for software product lines: A security domain requirements engineering process, Computer Standards & Interfaces 30 (6) (2008) 361–371.

[68] C. Trubiani, I. Meedeniya, V. Cortellessa, A. Aleti, L. Grunske, Model-based performance analysis of software architectures under uncertainty, in:

Proceedings of the International Conference on Quality of Software Architectures (QoSA), 2013, pp. 69–78.

[69] M. Kowal, I. Schaefer, M. Tribastone, Family-based performance analysis of variant-rich software systems, Software-engineering and management (2015).

[70] T. G. Kurtz, Solutions of ordinary differential equations as limits of pure jump Markov processes, Journal of applied Probability 7 (1) (1970) 49–58.

[71] S. Apel, A. Von Rhein, T. Thüm, C. Kästner, Feature-interaction detection based on feature-based specifications, Computer Networks 57 (12) (2013) 2399–2409.

[72] S. S. Kolesnikov, S. Apel, N. Siegmund, S. Sobernig, C. Kästner, S. Senkaya, Predicting quality attributes of software product lines using software and network measures and sampling, in: Proceedings of the International Workshop on Variability Modelling of Software-intensive Systems (VaMoS), 2013, pp. 6:1–6:5.

[73] N. Siegmund, M. Rosenmüller, M. Kuhlemann, C. Kästner, S. Apel, G. Saake, Spl conqueror: Toward optimization of non-functional properties in software product lines, Software Quality Journal 20 (3-4) (2012) 487–517.

[74] N. Siegmund, S. S. Kolesnikov, C. Kästner, S. Apel, D. Batory, M. Rosenmüller, G. Saake, Predicting performance via automated feature-interaction detection, in: Proceedings of the International Conference on Software Engineering (ICSE), 2012, pp. 167–177.

[75] G. G. Yen, Z. He, Performance metric ensemble for multiobjective evolutionary algorithms, IEEE Transactions on Evolutionary Computation 18 (1) (2013) 131–144.

[76] Y. Tian, R. Cheng, X. Zhang, Y. Jin, Platemo: A matlab platform for evolutionary multi-objective optimization [educational forum], IEEE Computational Intelligence Magazine 12 (4) (2017) 73–87.