June 30–July 4, 2014
Marcq-en-Baroeul, France

Association for Computing Machinery

Advancing Computing as a Science & Profession

QoSA'14

Proceedings of the 10th International ACM SIGSOFT Conference on

Quality of Software Architectures

(part of CompArch 2014)

Sponsored by:

ACM SIGSOFT

Association for
Computing Machinery

Advancing Computing as a Science & Profession

The Association for Computing Machinery
2 Penn Plaza, Suite 701
New York, New York 10121-0701

ISBN: 978-1-4503-2576-9

Printed in the USA

Table of Contents

Performance-based Selection of Software and Hardware Features under Parameter Uncertainty

Leire Etxeberria
Mondragon Unibertsitatea
Mondragon, Spain
letxeberria@mondragon.edu

Catia Trubiani
Gran Sasso Science Institute
L'Aquila, Italy
catia.trubiani@gssi.infn.it

Vittorio Cortellessa
University of L'Aquila
L'Aquila, Italy
vittorio.cortellessa@univaq.it

Goiuria Sagardui[*]
Mondragon Unibertsitatea
Mondragon, Spain
gsagardui@mondragon.edu

ABSTRACT

Configurable software systems allow stakeholders to derive variants by selecting software and/or hardware features. Performance analysis of feature-based systems has been of large interest in the last few years, however a major research challenge is still to conduct such analysis before achieving full knowledge of the system, namely under a certain degree of uncertainty. In this paper we present an approach to analyze the correlation between selection of features embedding uncertain parameters and system performance. In particular, we provide best and worst case performance bounds on the basis of selected features and, in cases of wide gaps among these bounds, we carry on a sensitivity analysis process aimed at taming the uncertainty of parameters. The application of our approach to a case study in the e-health domain demonstrates how to support stakeholders in the identification of system variants that meet performance requirements.

Categories and Subject Descriptors

D.2.11 [**Software Architectures**]; C.4 [**Performance of Systems**]: Modeling techniques, Performance Attributes; D.2.8 [**Software Engineering**]: Metrics—*performance measures*

Keywords

Software Architectures; Performance Analysis; Feature Selection; Uncertainty.

1. INTRODUCTION

In Software Product-Line (SPL) engineering, many software systems include a set of configuration options, called *features* [15, 1], that allow stakeholders to build system products fulfilling their functional requirements. Software and hardware features are also used to specify multiple architectural choices early in the design of a system [17], and these features aim to develop, deliver, and evolve a portfolio of similar products. However, in both cases, a major challenge is left in the hands of system designers that have to find an optimal product, i.e., the one that also meets non-functional requirements.

Performance is considered today a very critical non-functional property, because it directly affects user perception. If performance targets are not met, a variety of negative consequences (such as damaged customer relations, business failures, lost income, etc.) can impact on the project success [10]. All these factors motivate the activities of software performance modeling and analysis at the earlier phases of the life cycle, when reasoning on predictive quantitative results can allow to avoid expensive rework. However, early in the life cycle, there is often lack of knowledge about the software system, thus the performance analysis unavoidably co-exists with a certain amount of uncertainty.

In our previous work [22] we demonstrated that uncertainty is particularly critical in the performance domain when it relates to values of parameters such as the resource demand of services, service time of hardware devices, etc. In this paper we broaden the scope of our research, since we focus on studying the influence of uncertain parameters on system performance while selecting software (e.g., services) and hardware (e.g., single-core or multi-core processors) variable features. We achieve our goal by providing the performance prediction of best and worst cases on the basis of the selected features embedding uncertain parameters; thereafter, in case of wide gaps between worst and best case performance results (i.e., large uncertainty propagates from parameters to indices) we devise two sensitivity analysis steps aimed at taming the uncertainty of parameters.

Figure 1 illustrates the process we envisage for this goal. Ovals in the figure represent operational steps whereas square boxes represent input/output data.

We assume that a set of *Performance Requirements*, among others, is defined. Some examples of performance requirements are as follows: the response time of a service has to be less than 3 seconds, the throughput of a service has to be greater than 10 requests/second, the utilisation of a hardware device shall not be higher than 80%, etc. Performance requirements will be used to interpret the results from the model-based performance analysis.

[*]Leire and Goiuria are members of the embedded system group supported by the Basque Government.

QoSA'14, June 30–July 4, 2014, Marcq-en-Baroeul, France.
Copyright 2014 ACM 978-1-4503-2576-9/14/06 ...$15.00.
http://dx.doi.org/10.1145/2602576.2602585.

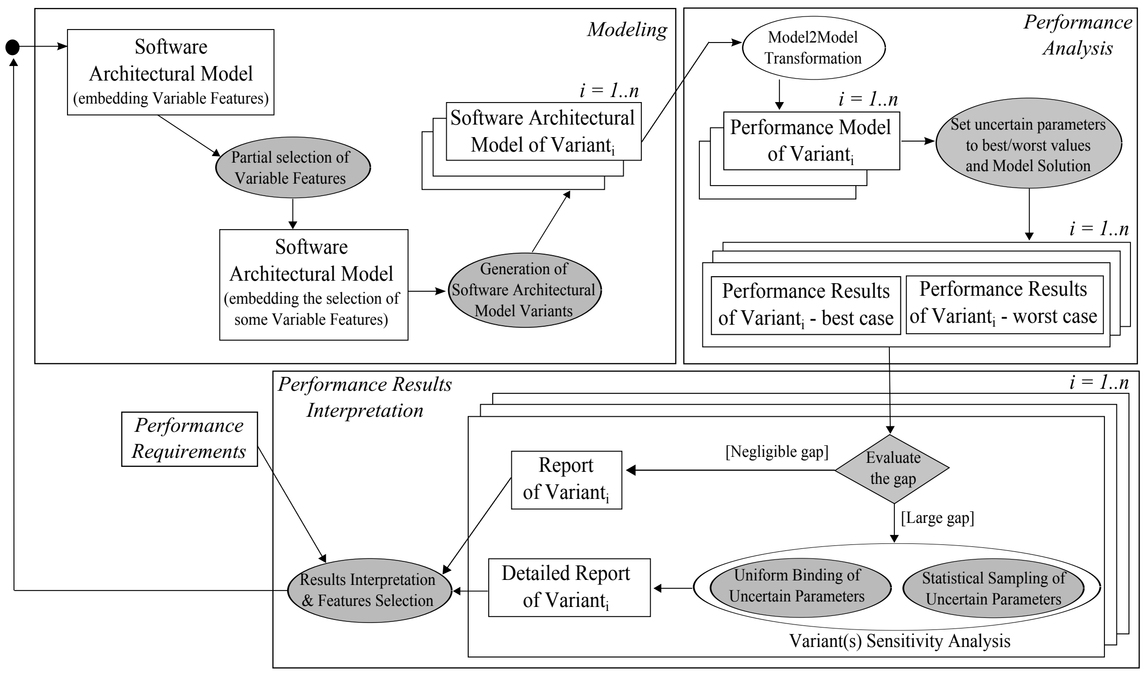

Modeling
Software Architectural Model (embedding Variable Features)
Partial selection of Variable Features
Software Architectural Model (embedding the selection of some Variable Features)
Generation of Software Architectural Model Variants
i = 1..n
Software Architectural Model of Variant$_i$
Performance Analysis
Model2Model Transformation
i = 1..n
Performance Model of Variant$_i$
Set uncertain parameters to best/worst values and Model Solution
i = 1..n
Performance Results of Variant$_i$ - best case
Performance Results of Variant$_i$ - worst case
Performance Results Interpretation
i = 1..n
Performance Requirements
Report of Variant$_i$
[Negligible gap]
Evaluate the gap
[Large gap]
Results Interpretation & Features Selection
Detailed Report of Variant$_i$
Uniform Binding of Uncertain Parameters
Statistical Sampling of Uncertain Parameters
Variant(s) Sensitivity Analysis

Figure 1: Model-based performance analysis process in presence of variable features embedding uncertain parameters.

Three macro phases have been identified in the process. In the *Modeling* phase, a software architectural model embedding variable features is built and, through the partial selection of variable features, a number of n software architectural model variants are generated. The modeling of variable features and the generation of variants are performed by using the SPL Conqueror tool [19]. In the *Performance Analysis* phase, we first transform software architectural models of the generated variants into performance models [3], then we set all uncertain parameters to their best and worst values, and at last we analyze the generated performance models. Note that we assume a property of monotonicity for the performance results with respect to uncertain parameters since the selected features of one variant (e.g., a system with four cores processors) only embed uncertain parameters (e.g., the service time of four cores processors may vary from 0.004 to 0.008 microseconds) that monotonically affect performance indices. Hence, the setting of uncertain parameters to best/worst values entails to solve the performance model of each variant twice and provides as output two different set of performance results for the best/worst case respectively. In the *Performance Results Interpretation* phase we evaluate, for each variant, the gap between performance results of best and worst cases. If the gap is negligible then a report of the variant is provided, otherwise we introduce two operational steps: (i) the uniform binding of uncertain parameters aims to reduce the uncertainty without further information, (ii) the statistical sampling of uncertain parameters aims to push ahead the uncertainty reduction given that parameter probability distributions are known. The comparison of results obtained by these two steps originates a detailed report of the variant under analysis. Finally, this report is provided to the system designers as support in the selection of software and hardware features that most likely fulfill the non-functional requirements. Integrating performance analysis into SPL in the early development phases allows to assess the impact of different choices on system performance. In case of unsatisfactory results, that is all variants do not fulfill performance requirements, a set of refactoring actions can be introduced to generate new software architectural models([1]) that undergo the same process shown in Figure 1.

The novelty of our approach is that it is able to quantitatively compare multiple system variants generated from the specification of variable features, thus supporting the software designer in the process of selecting the software and hardware features that most likely tolerate the uncertainty of parameters.

The remainder of the paper is organized as follows. Section 2 presents related work; Section 3 describes our approach; Section 4 shows the approach at work on a case study from the e-health domain; Section 5 reports the open issues raised by the approach; and finally Section 6 concludes the paper and provides directions for future research.

2. RELATED WORK

In this section we discuss related work in the context of variability-aware performance prediction that relates to two main research areas: (i) model-based performance analysis; (ii) optimization of non-functional properties.

Model-based performance analysis. In [21] a model-driven approach to derive a performance model from an extended feature model has been presented. Such approach is intended to integrate the performance analysis in the development process of Software Product Lines (SPLs), and provides performance predictions on the basis of product and platform-specific annotations. Our approach goes beyond feature selection because it keeps into account the uncertainty of parameters that triggers performance sensitivity analysis for system variants. In [9] a model-based approach

[1]We do not detail the refactoring process here, as it is out of this paper focus. However, readers interested to this part can refer to [2].

to estimate the performance of multi-core systems has been illustrated. It proposes a performance model for general-purpose operating system schedulers that allows the performance analysis taking into account the influence of schedulers in Symmetric Multi Processing (SMP) environments. We used this work to model the variability of operating system schedulers. In our previous work [4] a model-based analysis process for embedded SPLs has been presented, however it takes into account only variability issues and do not consider uncertainty in the parameters. We relied on this work for the modelling of variable software and hardware features, and we extend it by focusing on the model-based performance analysis of each variant.

Optimization of non-functional properties. In [19] the SPL Conqueror tool has been presented to optimize non-functional properties in SPLs. It generates the system variants, execute and measure it. In order to reduce the effort of analyzing all variants, the SPL Conqueror tool allows to get the minimum set of variants that is needed to calculate the impact on performance of each feature. SPL Conqueror [18] also considers feature interactions that affect non-functional properties and provides heuristics for their detection. In our approach we use this tool for the generation of software model variants. Recently, in [8] a variability-aware approach to performance prediction via statistical learning has been proposed. Such approach works progressively with random samples of variants, without additional effort to detect feature interactions. In [6] a contract-based technique is proposed to characterise worst case execution times under uncertain parameters, and such technique is applied to code examples. All these approaches are oriented to code-based measurement, since variants are executed to measure performance properties. On the contrary, our approach works on model-based performance analysis of variants and additionally considers uncertainty on parameters.

3. APPROACH

In this section we describe our approach by providing details on the shaded boxes of Figure 1 that represent the focus of this paper.

3.1 Partial selection of variable features and generation of software variants

A Software Product Line (SPL) is a set of similar software systems built from a shared set of features satisfying a particular domain, where a feature is defined as a prominent or distinctive user-visible characteristic of a system [12] [14]. An important challenge in the context of SPL is to manage the variability between products, where the variability is defined as the ability to customize a system [23] and also occurs when multiple architectural choices are allowed early in the design of a system [17].

The spectrum of variability may be quite wide, in fact a software system may include variability in the selection of software (e.g., service characteristics) and hardware (e.g., platform characteristics) features. A key concept in many SPL approaches is the feature model that represents the variability between family members in a concise taxonomic form. Hence, a specific system variant is characterized by a valid feature combination derived by the feature model based on the system's requirements.

In this paper we use feature models to represent software and hardware features that inevitably affect the software architecture as well as its performance. The selection of variable features is demanded to stakeholders, limitedly to the ones that are either optional or alternative. As a side effect, other features can be mandatorily required in the system configuration, once given the stakeholder's selection. This activity represent the first operational step in the *Modeling* phase (see Figure 1), and produces a software architectural model embedding the selection of some variable features.

Afterwards, the variability expressed in the feature model is analyzed and bound to specific system products that we name *variant*s in the following. Stakeholders perform an initial selection of features, then the list of variants (the minimum to get the impact of each feature) is automatically generated by SPL Conqueror tool. Software architectural models corresponding to such list of variants are generated automatically [4]. This activity represent the second operational step in the *Modeling* phase (see Figure 1), and produces a set of n software architectural models representing all the relevant system variants.

Note that there are some software and hardware features including uncertain parameters. For example, if a certain service (i.e., a software feature) is selected, then such service may include an uncertain parameter related to its resource demand (e.g., the processor units of computation needed to accomplish a software operation may vary from 10 to 15 no. of visits). Consider as another example if a four cores processor (i.e., a hardware feature) is selected, then such processor may include an uncertain parameter related to its service time (e.g., the service time of four cores processors may vary from 0.004 to 0.008 microseconds). On the other hand, there are some features that do not include uncertainty in their parameters. For example, if a fix allocation for multi-core processors (i.e., a hardware feature) is selected, then such allocation affect the routing probability among cores, but no uncertain parameters are introduced.

3.2 Set uncertain parameters to best/worst values and model solution

In the *Performance Analysis* phase (see Figure 1) each software architectural model representing a system variant is firstly transformed into a Queueing Network (QN) [13] performance model [3]. Since software architectural models contain a certain number of parameters (e.g., resource demand of services and service times of hardware devices) whose values are subject to lack of knowledge in the early phases of the software life cycle, this uncertainty is propagated by the transformation into performance models.

Hence, QN models contain a set of variable parameters that cannot be deterministically assigned. However, such parameters are bound to concrete values by estimating their best and worst values that could be available in many cases [7]. Hence, the setting of uncertain parameters to best/worst values entails to solve the performance model of each variant twice and provides as output two different set of performance results for the best/worst case respectively. In our approach the solution of QN performance models is performed with the Java Modelling Tool (JMT) [5].

It is worth to remark that each variant only embed uncertain parameters that monotonically affect performance indices. Hence, this step of model solution uniformly takes best and worst values for all the uncertain parameters thus to provide evidence of best/worst values for the indices.

3.3 Variants sensitivity analysis and performance results interpretation

In the *Performance Results Interpretation* phase (see Figure 1), the gap between worst and best cases is evaluated for each variant. Designers pre-define a percentage threshold on such gaps, possibly by interacting with stakeholders to interpret their level of uncertainty acceptance. The threshold is then used to distinguish between negligible and large (but not overwhelming) gaps. Negligible gaps do not lead to further analysis because the ranges of performance results of those variants have been considered as acceptable, and a variant report is produced. On the contrary, large (but not overwhelming) gaps trigger two performance sensitivity analysis steps driven by different criteria for dealing with uncertain parameters, as detailed in the following.

Uniform binding. It is used when the parameter probability distributions are unknown. Consequently, we aim at taming the uncertainty of parameters without additional information by assigning uniformly distributed values to all uncertain parameters. For this goal, we analyze the performance of each variant by splitting the parameter ranges in a fixed number of intervals and binding the parameters to the extreme values of such intervals, as formalized here below. Let m be the number of uncertain parameters, and let p_k be a generic assignment of values to these parameters, i.e., p_k is a vector of m elements $\{p\_k^j, 1 <= j <= m\}$. We name $worst_j$ and $best_j$ the worst and best values of j-th parameter, respectively. If n represents the fixed number of uniform intervals set for uncertain parameters, then each interval range is given by $(worst_j - best_j)/n$. Hence, p_k^j values will be defined as follows([2]):

$$p\_k^j = best_j + k * \frac{(worst_j - best_j)}{n} \quad (\forall k = 1, ..., n-1) \quad (1)$$

In other words, Equation 1 allows to calculate the value of j-th uncertain parameter related to the k-th assignment of values, where k varies in the interval (1, . . . , n-1) and n can be configured by the designer. As we show in Section 4, this step demonstrates that the propagation of parameter uncertainty on performance indices can be mitigated even in absence of further knowledge on the parameters.

Statistical sampling. It is used when the parameter probability distributions are known. Hence, we analyze the performance of each variant by sampling the values of uncertain parameters on the basis of their probability distributions. In our approach the sampling process is supported by ArcheOpteryx tool [16], where the number of samples can be configured by the designer.

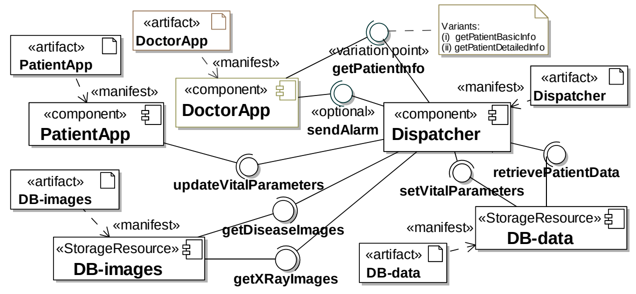
As we show in Section 4, an increased knowledge on parameters (even if still non-deterministic) allows to decrease the level of uncertainty on performance indices, thus driving software designers to a better feature selection than the best achievable one without this knowledge. This could be considered as an obvious assertion by a qualitative viewpoint that does not need any experimentation to be proven. However, the benefit of our approach is that it is able to quantify the propagation of uncertainty from parameters to performance indices. Hence, designers have in their hands a quantitative approach that can be appropriately tuned to drive their choices in different cases.

[2]In the formula it is assumed that worst values are larger then best values (e.g., resource demands, service time, etc.). In cases where it holds the opposite (e.g., network bandwidth), $worst_j$ and $best_j$ must be swapped.

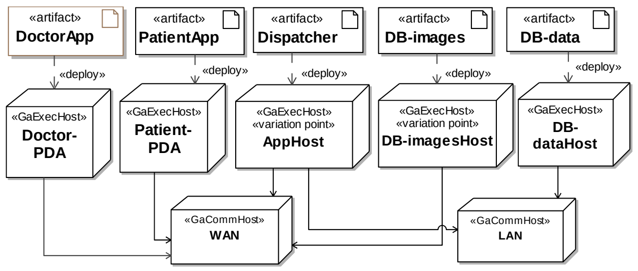
Both these two performance sensitivity analysis steps (i.e., uniform binding and statistical sampling) provide as output a detailed report for the system variant under analysis and it is used by the designers to select the software and hardware features that most likely fulfill the performance requirements.

4. CASE STUDY

The proposed approach has been applied on a case study in the e-health domain. Figure 2 depicts an excerpt of the E-Health System (EHS) software architectural model embedding variable features. The system supports the doctors' everyday activities, such as the retrieval of information of their patients. On the basis of such data doctors may send an alarm in case of warning conditions. Patients are allowed to retrieve information about the doctor expertise and update some vital parameters, e.g. heart rate, that are required to monitor their health status.



(a) Component Diagram.



(b) Deployment Diagram.

Figure 2: EHS- Software Architectural Model (embedding Variable Features).

The Component Diagram in Figure 2(a) describes the software components and their dependencies. *PatientApp* and *DoctorApp* components are connected to the *Dispatcher* component that forwards users' requests to the *DB-data* component and/or retrieves images from the *DB-images* component. The Deployment Diagram in Figure 2(b) shows that both the doctor's and the patient's applications are deployed on a Personal Digital Assistant (PDA), i.e. a mobile device in the hands of doctors and patients, respectively.

In the following we focus on *getPatientInfo* and *updateVitalParameters* services. When doctors require patient's information, their client application *DoctorApp* sends a request to the *Dispatcher* that manages the communication towards the database. Consequently, patient's medical history and disease data are retrieved by the *DB-data* storage resource, and if the battery of the doctor's PDA has a high charge level (regulated by a probability set to 0.75 in our case study) then the client application is notified by additionally sending x-rays and disease images retrieved by the

DB-images storage resource. When patients update their vital parameters, their client application *PatientApp* sends a request to the *Dispatcher* that manages the communication towards the database, and vital parameters are stored in the *DB-data* storage resource.

The system workload has been defined as follows: (i) a closed workload for the *getPatientInfo* service, with 50 users and an average thinking time of 5 minutes; (ii) a closed workload for the *updateVitalParameters* service, with 2500 users and an average thinking time of 1 hour. The performance requirements that we consider, under the stated workload of 50 doctors and 2500 patients, are:

RT: The average response time of the getPatientInfo service has to be less than 10 seconds.

RT: The average response time of the updateVitalParameters service has to be less than 5 seconds.

4.1 EHS Feature Modeling

Figure 3 reports the EHS feature model. *Hardware* features include the possibility to specify *multi-core* processors, with 2 or 4 cores for *AppHost* and *DB-imagesHost* (see Figure 2(b)). If a multi-core processor is selected, then *allocation* feature has to be defined. It can be fix or dynamic. In case of *fix* allocation it is possible to select the strategy to distribute classes of jobs: (i) Half and Half means that one half of the cores is dedicated to doctors' requests and the other half is dedicated to the patients' ones; (ii) One and Three can be selected in case of 4 cores only, and it means that one core is exclusively used by doctors' requests and three cores by patients' ones. In case of *dynamic* allocation we used the work in [9] to specify the variability in load-balancing policies of operating system schedulers. In our case, we devise two alternatives, that are: a more balanced one (i.e., 50%-50%) and a less balanced one (i.e., 80%-20%). In this latter case, doctors' and patients' requests are dynamically managed by an operating system balancing load among cores on the basis of the pre-defined probabilities. *Software* features include the possibility to specify different system functionalities (see Figure 2(a)). In EHS there are two alternative variants to get basic or detailed (i.e., doctors require the statistics of patients' vital parameters) patient information respectively. Furthermore, there is a service that can be optionally included in the system, i.e., doctors may send an alarm in case of warning conditions.

Table 1 schematically reports the output of SPL Conqueror tool on the EHS case study, given the feature model of Figure 3. After a partial selection where no multi-core is discarded, the tool provides a set of 7 system variants (over a set of 28 feasible variants) that need to be evaluated. As performance can only be measured per variant, SPL Conqueror automatically computes the minimum set of variants that include all the selected features (in this case 7), thus to get information about each feature. Note that $Variant_1$, $Variant_6$, and $Variant_7$ have the same hardware features, while they only differ for the set of provided functionalities. In fact, $Variant_1$ provides the *getPatientBasicInfo* service[3], $Variant_6$ provides the *getPatientDetailedInfo* service, and finally $Variant_7$ include the *sendAlarm* service. $Variant_2$ and $Variant_5$ are 4-cores configurations with a fix allocation: (i) half and half in case of $Variant_2$, (ii) one and three in case of $Variant_5$. Finally, $Variant_3$ and $Variant_4$ are 2-cores configurations with a dynamic allocation: (i) 50%-50% in case of $Variant_3$, (ii) 80%-20% in case of $Variant_4$.

[3]The *getPatientBasicInfo* service is shared by all the variants listed in Table 1, except for $Variant_6$ that provides the *getPatientDetailedInfo* service.

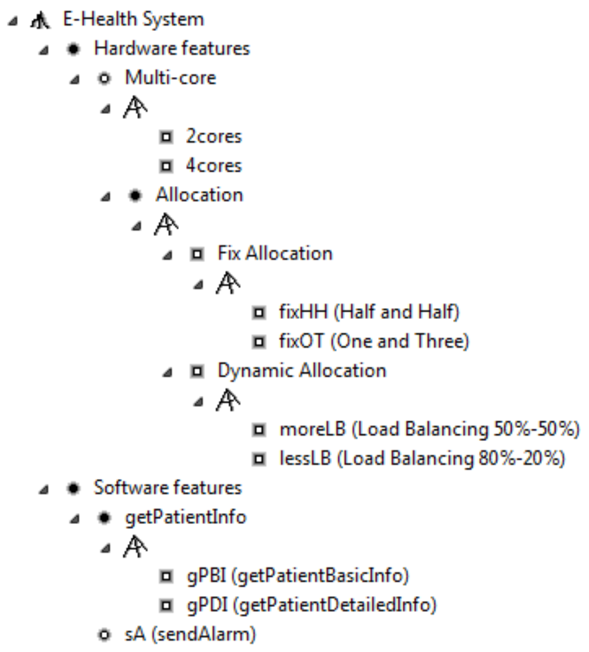


Figure 3: EHS- Feature model.

4.2 EHS Performance Analysis

Table 2 reports the EHS uncertain parameters, in particular five parameters concern the resource demands of software services, and three parameters are related to the characteristics of hardware devices. Each parameter is detailed with best/worst bounds as well as *mean* (μ) and *variance* (σ^2) of parameter probability distributions.

Table 2(a) reports the resource demand of software services expressed in terms of cpu visits each service requires to the host it is deployed on. Each parameter is defined with the name of the service and the host to which the demand is required. For example, the *getPatientBasicInfo* service has a parameter named *gPBI-AppHost* representing the number of visits to the *AppHost* that varies between *20* and *30*.

Table 2(b) reports the characteristics of hardware devices. State-of-art values [11] have been used to set these bounds: the service time of 2-cores processors varies between *0.007* and *0.014* microseconds, whereas it varies between *0.004* and *0.008* microseconds in case of 4-cores processors. Wide Area Networks (WAN) bandwidths have been estimated to vary between *6* and *2* Mbps.

As said in Section 3.2, the setting of uncertain parameters to best/worst values entails to solve the performance model of each variant twice. Table 3 summarizes the performance results of the EHS variants for the best/worst case respectively. We report the response time (RT) of critical services, i.e, *getPatientInfo* (namely *gPI*), *updateVitalParameters* (namely *uVP*), and *sendAlarm* (namely *sA*). Note that the response time of *getPatientInfo* service refers to *getPatientBasicInfo* in all variants, except in $Variant_6$ where such service is replaced by *getPatientDetailedInfo* (see Table 1). Similarly, the response time of *sendAlarm* service is only reported for $Variant_7$, in fact it is the only variant including such service (see Table 1).

	HW features						*SW features*	
	2cores	4cores	fixHH	fixOT	moreLB	lessLB	gPDI	sA
$Variant_1$	X		X					
$Variant_2$		X	X					
$Variant_3$	X				X			
$Variant_4$	X					X		
$Variant_5$		X		X				
$Variant_6$	X		X				X	
$Variant_7$	X		X					X

Table 1: EHS- model variants.

(a) Resource demand of *software* services.

Parameter	*Best value*	*Worst value*	*Mean (μ)*	*Variance (σ^2)*	*Unit of measure*
gPBI-AppHost	20	30	25	0.308	no. of visits
uVP-AppHost	80	120	100	4.938	no. of visits
gPDI-AppHost	5	7.5	6.25	0.019	no. of visits
sA-AppHost	10	15	12.5	0.077	no. of visits
gPBI-DB-imagesHost	40	60	50	1.234	no. of visits

(b) Characteristics of *hardware* devices.

Parameter	*Best value*	*Worst value*	*Mean (μ)*	*Variance (σ^2)*	*Unit of measure*
2coresProcessor	0.007	0.014	0.0105	1.51E-07	microseconds
4coresProcessor	0.004	0.008	0.006	4.94E-08	microseconds
WAN	6	2	4	0.049	Mbps

Table 2: EHS- uncertain parameters.

4.3 EHS Performance Results Interpretation

Table 3 additionally reports the percentage GAP we get in performance results for the best vs worst case, as the following ratio: $((worst - best)/worst) * 100$. For example, the value in $Variant_1$, $gap(gPI)$ cell states that the gap between worst and best cases for response time of *getPatientInfo* service in $Variant_1$ is 42.13%, and it has been obtained as $((3.94 - 2.28)/3.94) * 100$.

In our experimentation we focus on gaps from 60% to 90%, because the former value represents the acceptance threshold set by designers, whereas latter value guarantees to exclude unfeasible (i.e. too high) performance results that are usually originated by overstressing the system. For example, in $Variant_1$ the response time of *updateVitalParameters* varies from 2.36 to 600 seconds, where latter value unambiguously indicates the presence of a software/hardware bottleneck that delays all requests. $Variant_6$ and $Variant_7$ also suffer of the same problem. Variants that show such a behavior are straightforwardly discarded.

On the contrary, $Variant_2$ and $Variant_5$ are the best ones since the gap between best/worst values is lower than 50%, which is very likely due to their common feature to embed 4-cores processors that provide enough computational power to handle the worst case. Variants that show such a behavior nicely tolerate the uncertainty on parameters, hence designers do not strictly need any further analysis to deal with them.

Shaded entries in Table 3 highlight the variants that evidently need to be further analyzed by our sensitivity analysis process, because they show gaps within the previously defined thresholds. In particular, in $Variant_3$ the response time of *updateVitalParameters* varies from 2.15 to 6.15 seconds, whereas in $Variant_4$ the response time of *getPatientInfo* varies from 2.33 to 13.85 seconds. Note that both these latter variants do not fulfill the stated performance requirements in their worst case, hence designers are induced to discard them in the process of selecting features.

Figure 4 reports the sensitivity analysis of the *updateVitalParameters* response time in $Variant_3$.

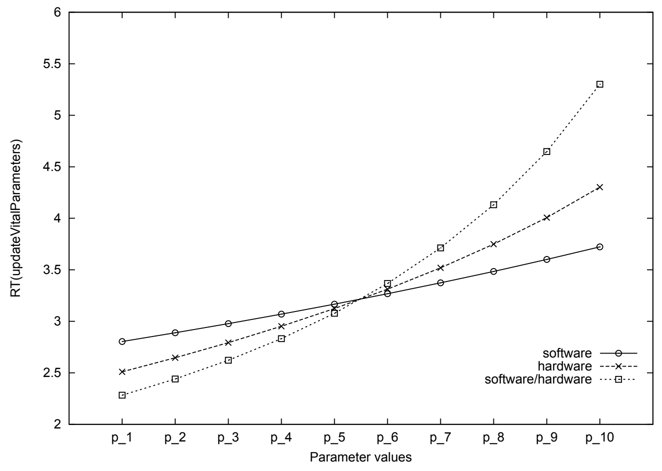
In particular, Figure 4(a) shows the variation of RT(updateVitalParameters) on the basis of parameter values that have been set according to the Equation 1 with n=11. We conducted three different experiments: (i) *software* parameters have been varied only, and hardware parameters are set to mean values; (ii) *hardware* parameters have been varied only, and software parameters are set to mean values; (iii) both software/hardware parameters have been varied. Note that n=11 leads to get 10 different sets of values for software, hardware, and software/hardware parameters, i.e., *p_1*, ..., *p_10* vectors reported on the x-axis of Figure 4(a). In particular, each *p_i* represents a set of values that have been calculated with Equation 1. This leads to a bench of 30 experiments. It is worth to notice that the increases of software uncertain parameters have little impact on the performance results since they lead the RT(updateVitalParameters) to vary from 2.80 to 3.72 seconds, whereas increases of hardware uncertain parameters lead the same index to vary from 2.51 to 4.3 seconds. Finally the variation of both software/hardware parameters has the highest impact on the performance results, as expected, since it leads the same index to vary from 2.28 to 5.30 seconds.

Figure 4(b) shows the response time histogram for these experiments, where the range of resulting RT(updateVitalParameters) values is reported on the x-axis and the number of occurrences on the y-axis. We remark that (5 + 7 + 5 + 5 =) 22 times the RT(updateVitalParameters) falls between 2.58 and 3.79 seconds, thus providing a confidence value for this interval of 0.73 (equals to 22/30).

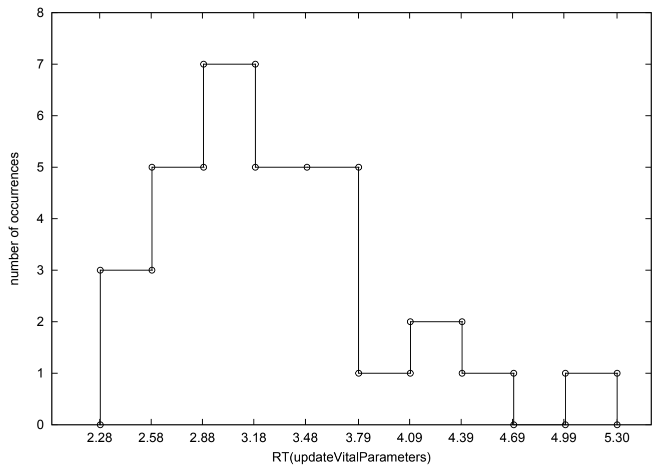
Figure 4(c) shows the variation of RT(updateVitalParameters) on the basis of parameter values that have been set by randomly sampling. We assumed that all parameters have a *NORMAL* distribution and, they have variances between $\mu - 3 * \sigma$ and $\mu + 3 * \sigma$ (see Table 2). This means that in 99.73% of observed cases the parameter values fall in

	RT (sec) - Best case			*RT(sec) - Worst case*			*GAP (%)*		
	RT(gPI)	RT(uVP)	RT(sA)	RT(gPI)	RT(uVP)	RT(sA)	gap(gPI)	gap(uVP)	gap(sA)
$Variant_1$	2.28	2.36		3.94	*600*		42.13	*99.61*	
$Variant_2$	2.09	1.8		3.42	3.21		38.89	43.93	
$Variant_3$	2.31	2.15		4.68	6.15		50.64	65.04	
$Variant_4$	2.33	2.23		13.85	*43.39*		83.18	*94.86*	
$Variant_5$	2.09	1.79		3.42	3.01		38.89	40.53	
$Variant_6$	3.18	2.37		6.47	*600*		50.85	*99.61*	
$Variant_7$	2.28	2.36	0.52	3.94	*600*	0.93	42.13	*99.61*	44.09

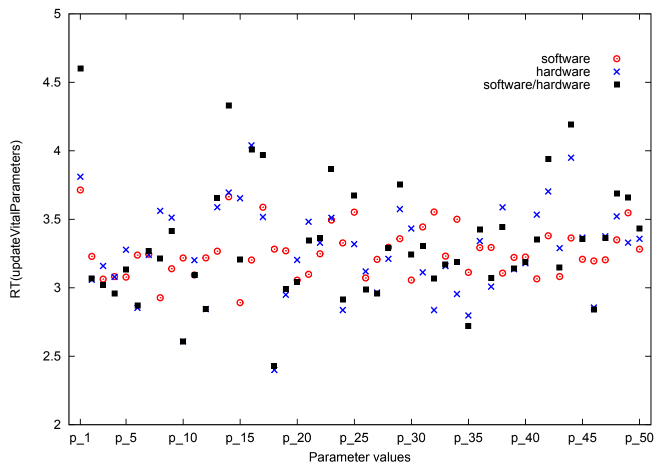
Table 3: EHS- RT of services while varying system variants (best and worst cases).



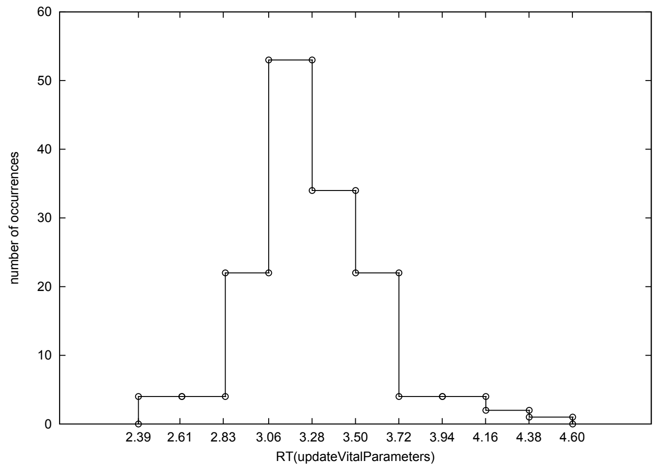
(a) Uniform binding: Response time vs software and/or hardware parameters.



(b) Uniform binding: Response time histogram.

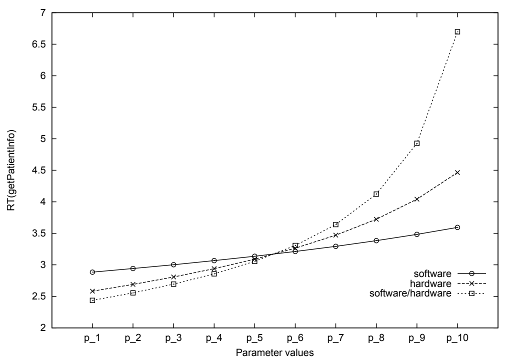


(c) Statistical sampling: Response time vs software and/or hardware parameters.

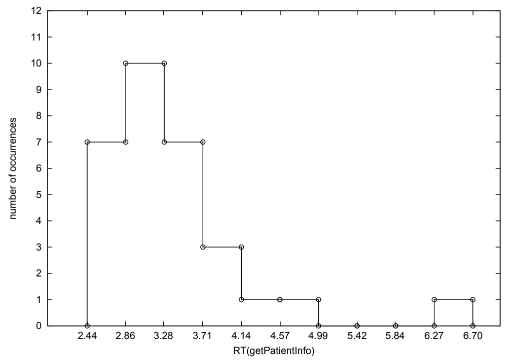


(d) Statistical sampling: Response time histogram.

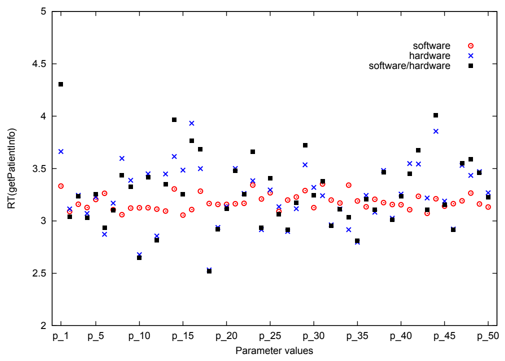
Figure 4: EHS-$Variant_3$: RT(updateVitalParameters) sensitivity analysis.



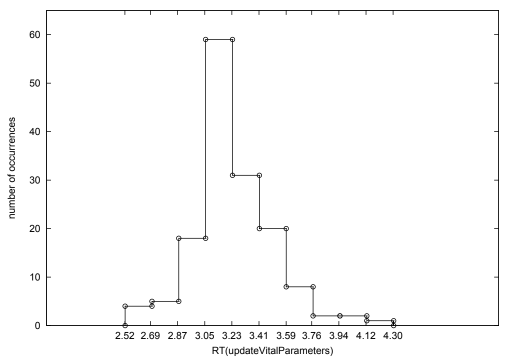
(a) Uniform binding: Response time vs software and/or hardware parameters.



(b) Uniform binding: Response time histogram.



(c) Statistical sampling: Response time vs software and/or hardware parameters.



(d) Statistical sampling: Response time histogram.

Figure 5: EHS-$Variant_4$: RT(getPatientInfo) sensitivity analysis.

the range of best and worst values. We conducted here the same three experiments shown in Figure 4(a). Note that 50 different values have been sampled for software, hardware, and software/hardware parameters, i.e., *p_1*, ..., *p_50* vectors reported on the x-axis of Figure 4(c). In particular, each *p_i* has been randomly generated by ArcheOpteryx tool [16], and the sampling process does not guarantee any order among the *p_i* vectors. This leads to a bench of 150 experiments. Even though the performance results seem to be scattered over larger ranges, we notice that the result trends in Figure 4(c) are similar to the ones obtained in the case of uniform binding. In particular, software uncertain parameters have little impact on the performance results since they lead the RT(updateVitalParameters) to vary between 2.89 and 3.71 seconds, whereas hardware uncertain parameters lead the RT(updateVitalParameters) to vary between 2.39 and 4.04 seconds. Finally the binding of both software/hardware parameters has more impact on the performance results, as expected, since it leads the RT(updateVitalParameters) to vary between 2.42 and 4.60 seconds.

Figure 4(d) shows the response time histogram for these experiments. We remark that (22 + 53 + 34 + 22 =) 131 times the RT(updateVitalParameters) falls between 2.83 and 3.72 seconds, thus providing a confidence value for this interval of 0.87 (equals to 131/150).

Summarizing, for $Variant_3$ the RT(updateVitalParameters) falls in the interval (2.58, 3.79) with a confidence value of 0.73, whereas in the interval (2.83, 3.72) with a confidence value of 0.87.

Figure 5 reports the sensitivity analysis of the *getPatientInfo* response time in $Variant_4$.

Similarly to Figure 4, Figure 5(a) refers to the uniform binding step, and it again shows that software parameters have little impact on the performance results, and so on towards concurrent variations of software and hardware parameters. Figure 5(b) shows the corresponding response time histogram that provides a confidence value of 0.8 (equals to 24/30) from 2.44 to 3.71 seconds. Figure 5(c) refers to the statistical sampling step, and it once again shows that software parameters have little impact on the performance results, and so on towards concurrent variations of software and hardware parameters. Figure 5(d) shows the corresponding response time histogram that provides a confidence value of 0.85 (equals to 128/150) from 2.87 to 3.59 seconds.

Summarizing, for $Variant_4$ the RT(getPatientInfo) falls in the interval (2.44, 3.71) with a confidence value of 0.8, whereas in the interval (2.87, 3.59) with a confidence value of 0.85.

The above experimentation confirms that, while increasing the knowledge on uncertain parameters, it is possible to better tame uncertainty in the performance results, and this was an expected finding. An added value of our approach is that it allows to quantify such progressive decrease of uncertainty, as demonstrated above. For example, we have shown that in $Variant_3$ the response time of the *updateVitalParameters* service falls in the (2.58, 3.79) seconds interval with a confidence value of 0.73 with the uniform binding methodology, whereas with the statistical sampling we demonstrated that it falls in the narrower (2.83, 3.72) seconds interval with a higher confidence value of 0.87. Other approaches that do not utilize uniform binding and/or statistical sampling can only report about best and worst cases, i.e. a (2.15, 6.15) seconds interval for the service here considered. This would negatively affect the stakeholders' choices in that, with a response time requirement of 5 seconds on the *updateVitalParameters* service, they would be induced to discard $Variant_3$ because the requirement falls within the (2.15, 6.15) seconds interval. Instead our approach is able to find out that such variant fulfills the requirement with a high confidence value.

Beyond the generation of variants, SPL Conqueror allows to automatically detect feature interactions. An interaction among two features occurs when the performance of a variant embedding both features are heavily different from those of the variant where one feature does not appear.

We remark that this aspect is not the focus of this paper, although we retained worthwhile to start investigating it. Thus, we used SPL Conqueror to detect feature interactions in EHS. We intended to investigate possible interactions between software and hardware features. This kind of interactions pointed out by the tool on EHS are: (2cores, moreLB, gPDI), (2cores, lessLB, gPDI), (4cores, moreLB, gPDI), (4cores, lessLB, gPDI)([4]). Hence, the focus here is on the interactions between number of cores (i.e., hardware) and *getPatientInfo* service behavior (i.e., software).

For sake of space we only consider here the following interactions: (2cores, moreLB, gPDI) and (2cores, lessLB, gPDI). These two sets of features give rise to new system variants, namely $Variant_8$ and $Variant_9$ respectively. Through explicit performance analysis of these variants we meant to validate the SPL Conqueror output.

To this end, Table 4 reports the response time of the *getPatientInfo* service, where shaded rows represent the two additional system variants. In the table $Variant_3$ and $Variant_4$ are also reported, because they differ from the additional ones by the absence of the *gPDI* feature. For each variant the response time of the *getPatientInfo* service has been obtained by assigning a random set of values to uncertain parameters, because in this experiment our intent was to focus on interactions rather than uncertainty. We have repeated this assignment several times, but for sake of space Table 4 only reports 5 different assignments named as *p_1*, *p_2*, . . . , *p_5*, one for each table column.

	RT(gPI)				
	p_1	*p_2*	*p_3*	*p_4*	*p_5*
$Variant_3$	3.24	2.73	2.83	3.29	3.00
$Variant_8$	6.18	4.56	5.20	6.24	4.94
$Variant_4$	3.69	2.85	3.06	3.88	3.23
$Variant_9$	12.30	5.57	7.71	13.62	6.46

Table 4: EHS- RT(getPatientInfo) service while considering two further system variants.

Numerical results provide good evidence to the correctness of SPL Conqueror output for these interactions. In fact, the system performance heavily vary, in both (2cores, moreLB) and (2cores, lessLB) cases, depending whether gPDI software feature is selected or not. For example, with the assignment *p_1* for uncertain parameters we can notice that the RT(gPI) delta percentage between $Variant_8$ and $Variant_3$ is equal to $(6.18 - 3.24)/6.18 = 0.47$, and the same quantity for $Variant_9$ and $Variant_4$ is $(12.30 - 3.69)/12.30 = 0.7$. The order of magnitude of these deltas has been observed across all adopted parameter assignments.

We are aware that this is far from being a rigorous proof of SPL Conqueror output soundness, but first validation results seem promising to track a direction for this goal.

5. DISCUSSION

Our approach highlights the complexity of model-based performance analysis of software architectures while considering software and hardware variable features as well as their uncertain parameters. In the following we discuss some key points raised by this work.

Software Product Line (SPL) principles. A remarkable difference exists between a software architectural model (embedding variable features) and a performance model. An architectural model describes a set of software products containing a collection of configurable software and hardware features, which are building blocks for many products with different options and alternatives. A performance model is an instance-based representation of one product at runtime, as deployed on a given platform. Thus, SPL principles do not apply to performance models since, as stated in [20], it is impossible to build a performance model for the whole set of products or to predict the performance properties of a software architectural model (embedding variable features). Hence, it is not possible to exploit the commonalities across variants in this context, but we need to instantiate a product in order to predict its performance characteristics (i.e., response time, throughput, utilization) on a specific platform.

Detection of feature interactions under parameter uncertainty. The detection of feature interactions may be affected by uncertain parameters since features including uncertain parameters (e.g., the multi-core processors) may propagate this uncertainty on performance indices, whereas other features (e.g., hardware allocation) do not. This point opens an interesting direction of research that we aim to investigate in the future.

Scalability of the approach. In our experimentation we analyzed 7 different QN performance models (that basically represent the system variants generated by SPL Conqueror), and each performance model has been solved twice to get best/worst cases. Two system variants were subject to sensitivity analysis, where the performance models were solved 30 times for the uniform binding, and 150 times for the statistical sampling. The JMT queueing network solver has evaluated a number of (7 * 2 + 2 * 30 + 2 * 150 =) *374* models in approximately 12 minutes, which can be considered more than acceptable at design time. In our future work we intend to investigate the scalability of our approach on larger QN models.

Limitations of the approach. Our approach to uncertainty taming only applies to parameters that monotonically affect performance indices. Consequently, there might be some parameters that cannot be considered by our approach because their influence on performance indices is non-monotonic. For example, the operational profile stochastically characterizes the way users interact with the system, hence it represents a good example of such a parameter.

[4]Note that each detected interaction is represented by a 3-ple because the multi-core feature (i.e., 2cores and 4cores) implies a choice on the allocation feature (i.e., moreLB and lessLB).

6. CONCLUSION

In this paper we proposed an approach to quantify the correlation between selection of features embedding uncertain parameters and system performance. In particular, we provided best and worst case performance bounds on the basis of selected features and, in cases of wide gaps among these bounds, we performed a sensitivity analysis process aimed at taming the uncertainty of parameters.

We have shown, through a case study, how taming uncertainty in performance results while increasing the amount of information available on uncertain parameters. Our approach allows to quantify the level of confidence that stakeholders can achieve on performance results, hence they can enhance their capabilities of feature selection by identifying system variants that better meet performance requirements.

As future work, we first intend to apply our approach to other case studies, possibly coming from real world systems. This broader experimentation will allow us to study the scalability of the approach on larger systems and compare the analysis results achieved on the model level with measurements from the actual system implementation. Beside this, we intend to integrate more sophisticated techniques to consider the variance in performance indices and study the interactions among features, as well as the influence of features on other non-functional attributes.

7. ACKNOWLEDGMENTS

The authors would like to thank Norbert Siegmund and Sven Apel (University of Passau) for their valuable support while using the SPL Conqueror tool. This work has been partially supported by the CRAFTERS ARTEMIS Project Nr. 295371, and the National funding Ministerio de industria, energía y turismo.

8. REFERENCES

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

[2] D. Arcelli, V. Cortellessa, and C. Trubiani. Antipattern-based model refactoring for software performance improvement. In *QoSA*, pages 33–42, 2012.

[3] S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni. Model-based performance prediction in software development: A survey. *IEEE Trans. Software Eng.*, 30(5):295–310, 2004.

[4] L. Belategi, G. Sagardui, L. Etxeberria, and M. Azanza. Embedded software product lines: domain and application engineering model-based analysis processes. *Journal of Software: Evolution and Process*, 2012.

[5] G. Casale and G. Serazzi. Quantitative system evaluation with java modeling tools. In *ICPE*, pages 449–454, 2011.

[6] J. Fredriksson, T. Nolte, M. Nolin, and H. Schmidt. Contract-based reusable worst-case execution time estimate. In *RTCSA*, pages 39–46, 2007.

[7] H. Groenda. Improving performance predictions by accounting for the accuracy of composed performance models. In *QoSA*, pages 111–116, 2012.

[8] J. Guo, K. Czarnecki, S. Apel, N. Siegmund, and A. Wasowski. Variability-Aware Performance Prediction: A Statistical Learning Approach. In *International Conference on Automated Software Engineering (ASE)*, pages 301–311, 2013.

[9] J. Happe, H. Groenda, M. Hauck, and R. H. Reussner. A prediction model for software performance in symmetric multiprocessing environments. In *QEST*, pages 59–68, 2010.

[10] H. Harreld. NASA Delays Satellite Launch After Finding Bugs in Software Program, April, 1998.

[11] J. L. Hennessy and D. A. Patterson. *Computer Architecture, A Quantitative Approach.* Elsevier, fourth edition, 2007.

[12] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical report, Software Engineering Institute, November 1990.

[13] E. Lazowska, J. Kahorjan, G. S. Graham, and K. Sevcik. *Quantitative System Performance: Computer System Analysis Using Queueing Network Models.* Prentice-Hall, Inc., 1984.

[14] K. Lee, K. C. Kang, and J. Lee. Concepts and guidelines of feature modeling for product line software engineering. In *International Conference on Software Reuse: Methods, Techniques, and Tools*, pages 62–77, 2002.

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

[16] I. Meedeniya, A. Aleti, and L. Grunske. Architecture-driven reliability optimization with uncertain model parameters. *Journal of Systems and Software*, 85(10):2340–2355, 2012.

[17] R. Olaechea, S. Stewart, K. Czarnecki, and D. Rayside. Modelling and multi-objective optimization of quality attributes in variability-rich software. In *NFPinDSML*, 2012.

[18] N. Siegmund, S. S. Kolesnikov, C. Kästner, S. Apel, D. S. Batory, M. Rosenmüller, and G. Saake. Predicting performance via automated feature-interaction detection. In *International Conference on Software Engineering (ICSE)*, pages 167–177, 2012.

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

[20] R. Tawhid and D. C. Petriu. Automatic derivation of a product performance model from a software product line model. In *SPLC*, pages 80–89, 2011.

[21] R. Tawhid and D. C. Petriu. User-friendly approach for handling performance parameters during predictive software performance engineering. In *ICPE*, pages 109–120, 2012.

[22] C. Trubiani, I. Meedeniya, V. Cortellessa, A. Aleti, and L. Grunske. Model-based performance analysis of software architectures under uncertainty. In *QoSA*, pages 69–78, 2013.

[23] J. van Gurp, J. Bosch, and M. Svahnberg. On the notion of variability in software product lines. In *Working IEEE/IFIP Conference on Software Architecture*, pages 45–54, 2001.