A Quality Model and Evaluation Method for Service-Oriented Software Product Line and Configurable Business Processes

BARDIA MOHABBATI, Simon Fraser University, Canada
DRAGAN GAŠEVIĆ, Simon Fraser University, Athabasca University, Canada
MAREK HATALA, Simon Fraser University, Canada
HAUSI A.MÜLLER, University of Victoria, Canada

Quality evaluation is a challenging task in monolithic software systems. It is even more complex and complicated when it comes to Service-Oriented Software Product Lines (SOSPL). In SOSPL, variability can be planned and managed at the architectural level to develop a software product with the same set of functionalities but different degrees of quality levels. A large variant space of both functional and non-functional (quality) properties provided by a product-line (or a family of services) complicates the measurement and optimization of quality aspects. Therefore, architectural quality evaluation becomes crucial due to the fact that it allows analyzing the potential of architecture to meet the required quality levels and validate whether the final product satisfies and guarantees all the ranges of quality requirements within the envisioned scope. This paper addresses the open research problem of aggregating quality ranges with respect to architectural variability. Previous solutions for quality aggregation do not consider architectural variability for composite services. Our approach introduces variability patterns that can possibly occur at the architectural level of an SOSPL. We propose a quality model for SOSPL and configurable business process models, and an aggregation method for quality range computation which takes both variability and composition patterns into account.

Categories and Subject Descriptors: D.2.4 [**Software Engineering**]: Reusable Software, Reuse models

General Terms: Design, Algorithms, Performance

Additional Key Words and Phrases: Service-Oriented Software Product Line, Service-Oriented Architecture, Quality-of-Service, Configurable Process Models

ACM Reference Format:
Bardia Mohabbati, Dragan Gašević, Marek Hatala, Hausi A. Müller, 2013. A Quality Model and Evaluation Method for Service-Oriented Software Product Line and Configurable Business Processes. *ACM Trans. Softw. Eng. Methodol.* , , Article (September 2013), 36 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

Service-Oriented Architecture (SOA) is becoming major paradigm to build versatile and distributed systems, that enables the realization of Software-as-a-Service. Although various stakeholders (e.g., companies and/or individual software developers) can employ services as the building blocks of their target application, in reality, they

This work is a revised and extended version of a paper published in Proceedings of the 9th International Conference on Service-Oriented Computing (ICSOC2011), Paphos, Cyprus, December 5-8, 2011.
Author's addresses: Bardia Mohabbati, Dragan Gašević, and Marek Hatala, Simon Fraser University, 13450 102 Ave. Surrey, Canada; emails: mohabbati,dgasevic,mhatala@sfu.ca; Dragan Gašević is also with Athabasca University, 1 University Drive Athabasca, Canada; and Hausi A. Müller, University of Victoria, STN CSC Victoria, P.O. Box 3055, Canada; email: hausi@cs.uvic.ca.

DOI: http://dx.doi.org/10.1145/0000000.0000000

often have dissimilar requirements from each other, resulting in the need for consideration of different features in the final product. This, along with the importance of sustaining market changes, necessitates mechanisms for the rapid development of software systems that best meet the stakeholders' feature needs. Some service providers have already moved towards the adoption of *customizable product-development models* (i.e., configurable business process models) to efficiently tailor solutions for their stakeholders. Within this process, they need to consider, manage and withstand both variable functional and non-functional (quality) requirements to systematically produce new applications. *Software Product Line Engineering (SPLE)* provides approaches to develop and maintain a family of products based on common aspects and predicted variabilities that allow rapid configuration and customization of new products. Several researchers have proposed to integrate SOA and SPLE paradigms into *Service-Oriented Software Product Lines (SOSPL)* as a way to achieve customizable product-development and take the benefits and synergies of both paradigms [Mohabbati et al. 2013b]. Researchers have explored various strategies for the realization of software applications, e.g., how the most appropriate services can be selected and how they can be efficiently composed for adaptable process models [Alrifai et al. 2012; La Rosa et al. 2011; Gottschalk et al. 2008; Cardellini et al. 2009; Ardagna and Pernici 2007; Zeng et al. 2004]. However, previous works often fails to consider Quality-of-Service (QoS) in the context of SOSPL and particularly for configurable business process models.

Form stakeholders' point of view, the quality characteristics (attributes) can be mandatory or optional with different levels of importance. Quality characteristics can be impact of variation in functional features. The influence of functional variable features on quality characteristics is complex and difficult to identify and measure. Because one variable feature of a service product family may be influenced by several quality properties required by stakeholder or one quality dimension may be affected by several variable features. It is necessary to identify and measure diverse impact of functional variable features on a quality attribute. These play a key role in assessing the quality characteristics for an SOSPL, which imposes heavy human effort for the evaluation.

Furthermore, to evaluate the potential quality of the developed service-oriented applications, it is essential to consider constraints and quality restrictions of individual services and overall requirements. Because some QoS dimensions values may vary during the service life-cycle. In consequence, the aggregation of quantified quality values is a prerequisite for composite service to assure and verify if the QoS expectations of stakeholder are satisfied.

Quality evaluation is a challenging task in monolithic software systems, and is even more complex when it comes to SOSPL, as it requires to analyze the quality characteristics of a family of SOA systems by considering variations in both functional and non-functional properties. Moreover, quality evaluation is an essential part of the optimization and configuration of product-line, because it provides a quantitative metric for the quality of the system based on architecture specification [Aleti et al. 2013]. In SOSPL, architectural quality evaluation becomes crucial as it allows the examination of whether the final service product satisfies and guarantees the ranges of quality requirements within the envisioned scope of the product line. This paper contributes a solution to the following open research problem: *How can the quantifiable values of quality dimensions of a service family be aggregated with respect to architectural variability?*

The novelty of our approach is in accounting for variability during architecture quality aggregation, which has not been considered in any related work, to the best of our knowledge. Our work focuses on the development of a framework for computing the quality ranges of features in an SOSPL by aggregating quality properties at the archi-

tectural level. This paper is revised and extended from its preliminary version [Mohabbati et al. 2011a] in which we proposed quality aggregation method based on structural variability and composition patterns in business processes models. We introduced a set of possible variability patterns that occur at the architectural level of an SOSPL. This can be seen as a catalog of patterns for variability may occur in the structure of business process models expressing composite services. Moreover, we provided the formalization of a computational model for architectural quality evaluation, which takes into account both variability and composition patterns and allows for trade-off analysis and architectural decision-making among options that provide similar functional properties but different quality levels. In particular, this paper makes the following additional contributions:

(1) We introduced *an extensible multidimensional QoS model* to captures non-functional properties that are inherent to an SOSPL.
(2) We developed a *quality model framework* for holistic architectural quality evaluation based on different architectural patterns.
(3) We conducted sets of experiments based on real-data and synthesized data sets for performance analysis of proposed QoS aggregation method.

The rest of this paper is organized as follow: Section 2 describes related conceptual modeling and formalism for configurable business process model in the context of SOSPL. The proposed QoS model is reviewed in Section 3. Section 4 describes and exemplifies the QoS aggregation and computation method. The evaluation of method is presented in Section 5. The related work is discussed in Section 6. Finally, Section 7 presents the conclusion and future work.

2. SERVICE-ORIENTED SOFTWARE PRODUCT LINES AND CONFIGURABLE BUSINESS PROCESSES

SPLE has been recognized as a successful approach for the variability management and reuse engineering, which enables mass customization, enhances software quality and reduces the time-to-market of new software products [Pohl et al. 2005]. Different software products derived from a software product line are distinguishable based on their included features. A *feature* reflects the stakeholders' requirements. It is an increment in the product functionality and offers a configuration option [Pohl et al. 2005]. Given this definition for a feature, SPLE relies on the essential concepts of *commonality* and *variability* of features among products.

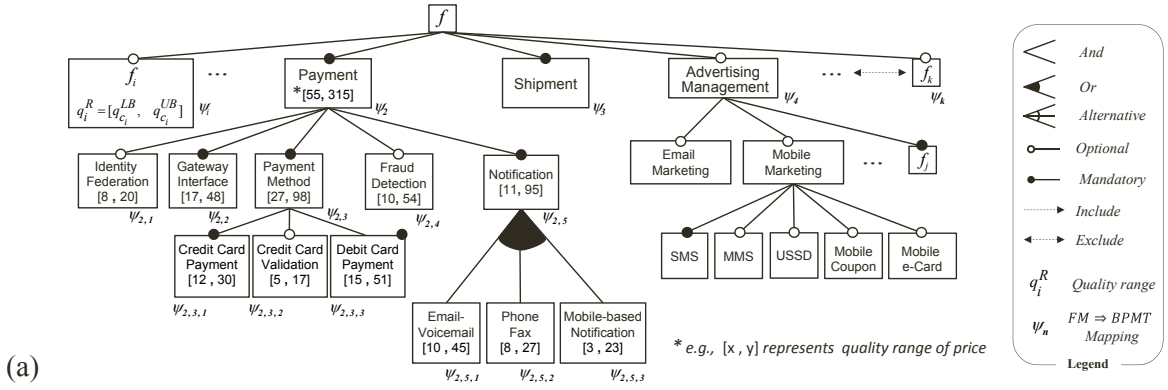
SPLE consists of two main lifecycles: *Domain Engineering* and *Application Engineering* [Pohl et al. 2005]. Domain Engineering is concerned with the analysis and identification of the scope of the product line and the capturing of the entire domain of interest through modeling of common and variation points. An Application Engineering cycle builds the understanding of specific requirements of different stakeholders, for whom the customization and configuration of the product line is carried out. We have presented the details of these two distinctive lifecycles in [2013a; 2011b].

Given that software product line models are often abstract representations of a domain/application of interest, it is important that they are interrelated with solution space models that would allow their actual operationalization. To this end, many researchers and practitioners have already investigated the importance of leveraging the synergies between SOA and SPL to create Service-Oriented Software Product Lines (SOSPLs) [Mohabbati et al. 2013b; Cohen and Robert 2010; Lee and Kotonya 2010]. Such approaches benefit from SOA principles to provide an actual implementation of SPL products. Accordingly, SPLE helps to develop variant-rich service products and enable reuse and optimization with business process through commonality and variability management. Let us review an illustrative example in this regard.

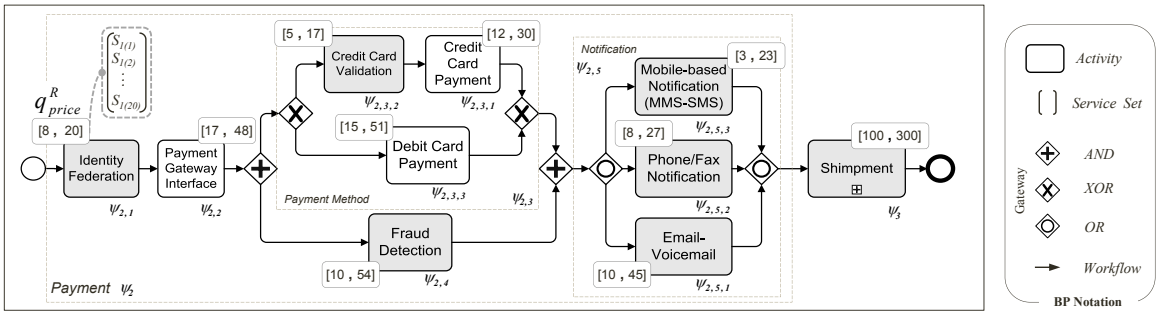
2.1. Illustrative Example

We focuses on the simplified business process model for payment service of our case study in a global online retailer scenario to illustrate the concepts and further discussions in this paper. Figure 1 (a) illusterates a feautre model representing commonalities and variabilities in a configurable process model of payment service (See Figure 1(b)). The feature model focuses on the structural relationships and the configuration dependencies between features, and it is used to distinguish products of a product line of services and guide the configuration of a reference process model.

As shown, different features from the feature model (on the top) can be used within the business process model (on the bottom) to provide different functionalities of the payment service as a composite service. These features can be realized using appropriate services, which can have different QoS characteristics. The activities represented in gray color in the process model indicate optional features; i.e., those features can be optionally included or excluded from the target product based on stakeholders' requirements. For instance, the `Notification` feature can be included in a final service-product by selecting one of the `Mobile-based notification`, `Phone-Fax notification`, or `Email/Voicemail notification` features, which have different range of quality values, since they are implemented by different services. Hence, the QoS characteristics of a developed product are closely dependent upon the features that get included in a final product.



(a)



(b)

Fig. 1. a) Feature model representing structural variability in business process family b) payment feature and its process model.

A feature model such as the one in Figure1 (a) is a model of a family of payment process, which in turn represents a configurable SaaS, while each variant (configured service) is a member of that family. As we discussed, variation points are considered those places in the design of a configurable process model. However, the options to be selected for a particular application w.r.t. stakeholders' requirements are left open for the configuration. Hence, variation points provide the possibility to derive different products, i.e., different final composite services. In SOSPL, particularly during the domain engineering cycle, determining the implied QoS ranges for individual features, based on the underlying architecture and implementation, helps domain engineers ensure that the product line architecture will fulfill and deliver the upper and lower bounds or values of quality requirements requested by stakeholders. In other words, the aggregation of the QoS characteristics of a feature model based on the quality of its features, as derived from underlying processes and services implementing those features, provides the means to estimate the likely lower and upper bounds of QoS properties for potential service products that will be derived from that product family. In addition, in the context of SOSPL, quality range computation through the construction of a generic evaluation model enables us to keep track of the product line quality ranges even during or after specifications of the service quality have changed. In next sections, we describe our proposed approach how these QoS ranges are computed in the presence of variability.

2.2. Reference and Configurable Process Model

Today, software products as services are designed, built, executed, maintained and evolved by means of business processes on top of Web services technologies [Dumas et al. 2012; Ouyang et al. 2009]. A template-based approach, where a reference model is designed as a template and is further configured or customized for various purposes, has been widely adopted by practitioners [Czarnecki 2005]. for the entire product line in a superimposed way. The reference model provides the common business logic for orchestration and choreography of services.

The configuration (tailoring and customization) of a reference models is performed by the selection/elimination of features from the feature model. In other words, due to the fact that architectural variations in the reference model are encoded as features, the various parts of the reference business processes are organized into variation points, which are managed and configured by means of feature models. Feature models capture and encapsulate only architectural variability at design time. In contrast, business process models describe behavioral variability, i.e., how features are composed, which drives runtime variability through composition patterns (discussed in the next section).

As mentioned earlier, complex software application can be build out of composite services. A service can be modeled as a software component with a well-defined interface that implements a set of operations offering piece of functionality. A (business) process model describes *composition logic* defining workflow (i.e., control flow) of a set of ordered activities and their transitions (also called service activity) which realize and implement features of software w.r.t. stakeholders requirements model. It enables services to be composed at the different levels of granularity and dictates how services can be combined, synchronized, and co-ordinated.

We refer to *reference process model* as an abstract representation of all valid service compositions in a family of business processes. We assume hat a reference process model, as a higher level of compositions, is generated by template-based and parametric-design-based approaches. A reference process model PM consists of a set

of interrelated abstract activities $\mathcal{A} = \{a_1, \ldots, a_n\}$ and their data dependencies (inputs, outputs, pre- and post-conditions (i.e., effects) according to regarded workflow structures. An activity a_j represents a well-defined business function, as a functional abstraction of a service providing implementation (i.e., operationalization) for feature f_j. Hence, an activity can be atomic (e.g., atomic service) or non-atomic (i.e., composite service). Each activity is delegated to one or more *concrete services* providing the required functionality with different range of quality, e.g., in terms of cost and performance. We formally define a process model as follows:

Definition 2.1 (*Process Model*). A process model *PM* is modeled as a directed acyclic graph $G_{PM} = (V, E)$, where $V = \{V_\epsilon, V_\sigma, V_A, V_G\}$ denotes a set of disjoint nodes, V_ϵ and V_σ are set of initial and final nodes, respectively. V_A is a set of activities, and $\mathcal{E} \subseteq V \times V$ represents the workflow (transitions) between the nodes. V_G is a set of nodes as gateways. A gateway has a type $T(V_G)$ such that $T(V_G) \in \{$*AND, OR, XOR, DISC*$\}$.

Gateways, as routing constructs, represent a workflow of branching (i.e., routing points), and workflow patterns [Van Der Aalst et al. 2003] describe the structure and behavior of processes for the execution. Workflow patterns are defined in terms of how the process flow proceeds in sequences and splits into branches to execute the activities and how they converge.

To model and support peculiar characteristics of process models and stakeholder's requirements, the specifications of process model are enriched by *semantic annotations*. We assume the execution probability of every conditional branching i.e., switch (XOR-split), are specified by annotations. Therefore, for each gateway with conditional branching with k disjoint branches $\sum_{i=1}^{k} \rho_i^b = 1$, where ρ_i^b indicates the probability of execution of ith branch. Furthermore, loop constraints in a process model can be defined to express the number of iterations for particular activity. Hence, the probability distribution of every loop with maximum number of iterations c is specified such that $\sum_{i=0}^{c} \rho_i^l = 1$, where ρ_i^l indicates the probability that loop l executed is i times. We assume there is an upper bound c for the loop l is determined. Otherwise, the process further cannot be optimized because infinitive resources might be required for execution. The distribution of probabilities of execution of conditional branches and loops are specified during design-time or available and evaluated from past executions and captured by systems logs or service brokers [Zeng et al. 2004].

To describe a set of functionally equivalent services with different QoS, we define service candidate set denoted as $\mathcal{S}_{a_i}$ for each activity a_i implementing and executing feature f_i. Given the above consideration, a set of $\bigcup_{i=1}^{n} \mathcal{S}_{a_i}$ includes binding links to services providing multiple implementations for n activities in a reference process model. The best services for each activity is further selected by optimization process according to required QoS constraints specified by stakeholder. Selected services are invoked at run-time by dynamic/late binding mechanism. The information about services is managed by a service broker.

In this work, we considered *structured process models* which have a number of desirable properties [Polyvyanyy 2012; Kiepuszewski et al. 2000]. A process model is called to be structured if it has following properties: (1) the soundness property of process model can be checked in polynomial time. A process model with no structural errors such as deadlocks or lack of synchronization [Sadiq and Orlowska 2000] is called to be sound [Aalst et al. 2002], and (2) it has the properties of modularity, readability and maintainability. The soundness of a structured process model can be analyzed by Petri net reachability graphs.

Process Structure Tree: To allow further analysis of activities and their relationships according to workflow patterns, we rely on the concept of *Process Structure Tree*

(PST). This concept is based on partitioning a process model into smaller fragments (regions or components) and structuring them in a hierarchical way according to the nesting relation. PSTs can be created using different algorithms. Our approach is based on method described in [Vanhatalo et al. 2007], which decompose process model into canonical *Single-Entry-Single-Exit (SESE)* regions. SESE regions are known from compiler theory and are utilized for control flow decompilation.

We consider (well-) structureness property of a process model [Polyvyanyy 2012; Kiepuszewski et al. 2000]. A process model is well-structured if and only if for every node with multiple outgoing arcs (i.e., a split), there is a corresponding node with multiple incoming arcs (a join), such that set of nodes between the split and the join forms SESE region. An structured process model can be decomposed into SESE regions [Vanhatalo et al. 2009], which can be computed in linear time (cf. [Johnson et al. 1994]). SESE regions are defined formally as as follow:

Definition 2.2 (*Single-Entry-Single-Exit Region*). Let G_{PM} be a process model graph with distinguished initial node and final node, such that every node is reachable from initial node and final node is reachable from every node in G_{BP}. Two distinct nodes V_i and V_j in G_{BP} enclose a *Single-Entry-Single-Exit (SESE)* regions if

(1) V_i *dominates* V_j, i.e., every path from initial node to V_j includes V_i
(2) V_j *postdominates*, i.e., every path from V_j to final include V_i
(3) every cycle containing V_i also contains V_j

We define *Process Structure Tree (PST)* as a hierarchical representation of a structured process model according to Definition 2.1 and 2.2.

Definition 2.3 (*Process Structure Tree*). Given a process model PM, $PST_{PM}=(V,E)$ is a tree of canonical fragments of G_{PM}, where $V=\{V_r,V_R\}$ is a finite set of nodes. V_r is the root of tree. V_R corresponds to canonical SESE regions. $E \subseteq (V_R \times (V_R \backslash V_r)\}$ is the set of edges. Nodes are either activities V_A or gateways V_G.

2.3. Feature Model and Variability Patterns

As we mentioned earlier, feature model prolongs to support configuration framework in order to support dynamic deployment of variant business processes or enable *configurable business process*. A feature model is a means for describing a permissible configuration space of all the products of a service family in terms of its features and their relationships. Figure 1(a) depicts a part of the feature model in our example, which breaks down the variability and commonality of a product line of services into a hierarchy of features and represents structural variability in the reference process model. Parent-child relationships in the feature diagram indicate the refinement of application functionality. As not all features are assumed to be present in every product, this differentiation is expressed by a classification of feature types and relationships, which drive *structural variability patterns* as follows:

— **Mandatory/Optional:** A mandatory feature must be included in every member of a product line of services if its parent feature is selected. An optional feature may or may not be included if their parent is included;
— **Or-group:** Or-feature group is non-empty subsets of features that can be included if a parent feature is included;
— **Alternative-group:** Alternative-feature group indicates that from a set of alternative features exactly one feature must be included if the parent of that set is included.

Variation points are those features that have at least one direct variable sub-feature (See Figure 1). It can be observed, except for the mandatory feature type, all the other types of features imply structural (architectural) variability.

A feature model can be expressed as a rooted directed acyclic graph (DAG). Feature diagram, which is a tree-like graphical notation, is more widely used due to its visual appeal and easier understanding. In the tree-like structure, each node (feature) represents a variation or increment in application functionality. We formally define a feature model as follows:

Definition 2.4 (***Feature Model***). A feature model FM is a directed acyclic graph $G_{FM}= (V_F, r, E, \lambda,)$ where

— $V_F = \{f_1, \ldots, f_n\}$ is a finite set of nodes representing features and theirs attributes;
— $r \in V_F$ is the root feature of feature model;
— $DE \subseteq N \times N$ is a set of decomposition edges representing the parent-child relations;
— $\lambda : \mathcal{P}(f) \rightarrow NT \subseteq N \times N$ is a function assigning feature types and cardinalities for child features with parent feature f, where $NT \in \{\overset{\bullet}{f}, \overset{\circ}{f}, f_{or}, f_{xor}\}$. The $\overset{\bullet}{f}$ and $\overset{\circ}{f}$ denote mandatory and optional parent-child feature relations, respectively; f_{or} and f_{xor} denote Or and Alternative group relations between parent-child features with common parents, respectively;
— $CE \subseteq N \times N$ is a set of constraints edges expressing the integrity constraints among features.

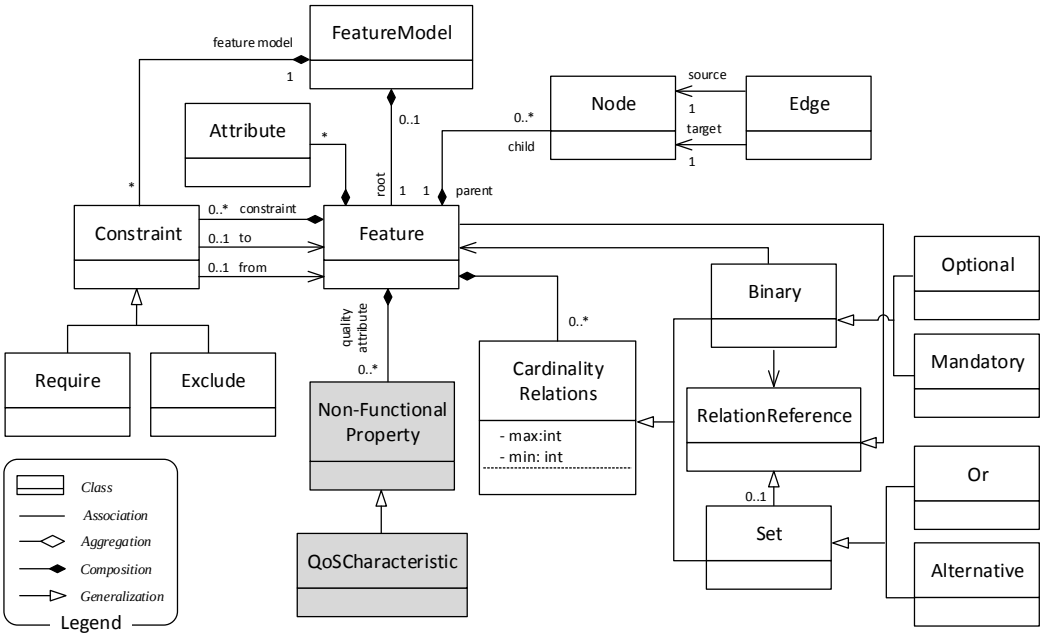


Fig. 2. Meta model of extended cardinality-based feature model.

Figure 2 shows the meta model of cardinality-based feature model and the major constructs. A feature model with a root feature is composed by an optional set of relations. The cardinality-based relations can be two distinctive types: “binary” relations

expressing optional and mandatory features, and "set" relations constructing grouped features.

In general, *cardinality* specifies reasonable interval for number of child-feature in a group of features $F \subseteq N \times N$, which is denoted by $<k-k'>$, where $1 \leq k \leq k' \leq m$, and $m = |F|$. Hence, $<k-k'>$ cardinality defined over *Or feature groups* indicates at least k and at most k' features that can be included out of the m features in a group if the parent is selected. Moreover, the $<k-k>$ cardinality specified for *Alternative feature groups* implies that only k out of m features in the group must be included if the parent is selected. The concept of cardinality also emulates four types of decomposition for features in a group as follows: $<m-m>$, $<0-m>$, $<1-m>$, and $<1-1>$ specifies the mandatory, optional, Or, and Alternative features/groups, respectively.

A feature and corresponding service may depend on other features for its correct functioning or operation. Therefore, operational dependencies among features can be defined by constraints which are known as *integrity constraints* (or cross-tree constraints) [Lee and Kang 2004]. For instance, a feature can refer to another feature that it *requires* (*includes*) or *excludes*. Therefore, the presence of a certain feature in the product may impose the presence or exclusion of another feature. Integrity constraints can be expresed and specified by the constraints edges in a feature model. Thereby, service designer can express implicitly or explicitly dependency among features.

We extendeded the conventional feature model to incorporate *non-functional properties* which can be further defined as qualitative or quantitative quality characteristics (See Figure 2). Thereby, each feature can comprise properties to include quality information. In SOA, non-functional properties can be viewed as *QoS characteristics*. In the next section we describe our proposed QoS model which describes quality characteristics and extends feature model.

3. QUALITY OF SERVICE MODEL FOR A SERVICE-ORIENTED SOFTWARE PRODUCT LINE ARCHITECTURE

This section introduces a QoS model and also provide the foundation for QoS aggregation and computation, and further quality-aware SOSPL configuration.

3.1. Quality of Service Requirements and Criteria

In contrast to functional properties defining particular function or behaviour of a software system, non-functional (also known as extra-functional or quality) properties describe how well functionality of system is fulfilled. We discriminate between two general types of quality characteristic for a service-oriented software product line (service family): (1) the quality attributes which are specific to the product-line and are inherent to undertake a set of related products as we all new feature in terms of a service. Such those quality variability characteristics are related to functional variability causing variation in the quality of product-line members, and (2) the general quality or domain-specific quality characteristics, i.e., cost, performance, cost, safety, etc. We only consider general quality type for further quality aggregation and computation in this paper.

Existing standards on software product quality (e.g., ISO/IEC 9126-1 and updated version ISO/IEC 25010 [2010]) provide insight into general and common characteristics for almost every type of software. However, different types of software products have specific characteristics. Hence, the actual application of software quality models usually requires reusing an existing quality model and extending it for a specific software product or domain. In SOC community, the "Quality of Service", or QoS for short, are referred to as non-functional properties associated with a service, which are measured to evaluate the degree a service meets quality requested by stakehold-

ers [Menascé 2002]. These qualities are described in a Service Level Agreement (SLA) serving as the foundation for expressing non-functional requirements and the expected level of service between the consumer and provider.

The importance of QoS for SOA is due to the fact that service-oriented applications often use highly-distributed and loosely-coupled services over the network. These services may be invoked and executed by a variety of stakeholders in heterogeneous environments. Accordingly, the modeling, describing and managing service quality are of utmost importance in all the SaaS life-cycle phases, requirements specification to design, deployment and execution monitoring.

3.2. QoS Model

The QOS model is utilized to define quality specifications and make explicit various characteristics and dimensions that specify the expectations on quality levels provided by service. The QoS model is used by service requesters (i.e., stakeholder) to express QoS requirements and preferences and by service providers to describe and advertise services QoS, determine the priority and relationships among QoS aspects that are used for further reasoning, decision-making, and service configuration and optimization.

In this section, we describe our proposed QoS model by following ontology-based approach which enables us to express QoS required for a SOSPL.

To end this, we firstly introduce primary general requirements and design criteria guiding to develop a QoS model. An extended review of number of existing QoS languages [Vedamuthu et al. 2007; Andrieux et al. 2005; Paschke 2005; Tosic et al. 2005; Ludwig et al. 2003], ontologies [Tran et al. 2009; Papaioannou et al. 2006; Kritikos and Plexousakis 2006; Ran 2003], and models [Jureta et al. 2009; Herssens et al. 2008; Toma et al. 2006; Wang et al. 2006; O'Sullivan et al. 2002] specified these general design requirements which is also summarized by Tran et al. [Tran et al. 2009] as follow. A QoS model should enable to conceptualize and express explicit specifications of QoS properties that can be applicable for various application domain. In addition, it should be extensible to define and include arbitrary QoS properties w.r.t. domain of interest. QoS model should support different participants (e.g., provider, stakeholder, third-parties, brokers, etc.) specifying quality information in details and requirements at different level of expectations. Quality information can be applied to different elements of an atomic or composite service (e.g., service operation, input/output, and interface). For product-line engineering, different types of quality variability can be identified: (a) variability among different quality properties, (b) different priority levels or preference on quality properties, and (c) indirect variation which is resulted in functional variability. Therefore, the QoS model should support to express stakeholder preferences, priority and conditions concerning QoS properties (e.g., mandatory and optional properties). Also, it should allow to describe the classification, relation and prioritization of quality aspects.

Our QoS model is based on the OMG UML QoS profile [SOC 2008] and extended version proposed by [Jureta et al. 2009], which conceptualizes various aspect of quality and enables the definition of standard QoS modeling elements. We also considered existing QoS ontologies [Tran et al. 2009; Papaioannou et al. 2006; Kritikos and Plexousakis 2006; Maximilien and Singh 2004; Ran 2003]. The OMG UML profile and existing ontologies don not take preference into account to express the relative importance of QoS for systems and stakeholders, which is perquisite and used for decision-making and reasoning over QoS properties and requirements.

The two important and conspicuous characteristics of our proposed QoS model are as follow: (1) it enables to define general and domain-specific quality aspects with their relations and allows to manage and group complex quality characteristics; and (2) the

model integrates the preference model that further enables to specify the quality relative importance and preferential aspects required for multi-criteria decision-making and automatic configuration of service. In the following, we first overview the key concepts of the UML QoS profile (the details of meta model is available in [SOC 2008], and then we describe the extensions to the QoS meta model by accommodating the preference model with the aims at supporting decision-making and optimization in the course of process configuration, customization and service selection.

To describe quality concepts, the UML QoS meta model introduced by OMG comprises three main conceptual submodels: Characteristics, Constraints, and Levels. Figure 3 shows the key elements of this meta model which is further instantiated to obtain a QoS model. The quality characteristic submodel integrates the concept of quality dimensions which are instantiated to represent quantifiable characteristics.

(1) QoSCharacteristic includes the model elements for the description of QoS properties representing a measurable non-functional aspect of a service within a given domain, for instance, *price*, *availability*, *throughput*, *scalability*, *accuracy*, and *reliability*. The Parent-Sub provides support for the specialization and extension of such elements. The template-derivation defines a relationship which enables a characteristic can be derived into other characteristics.
   - QoSParameter provides parameterization methods to quantify QoS characteristics in terms of units and types for the description of value definitions.
   - QoSDimension defines dimensions to quantify a quality characteristic. A QoS characteristics can be quantified and composed by multiple quality dimensions, and a quality dimension of QoS characteristics can b be defined based on another QoS characteristics. For instance, *reliability* characteristic can be an aggregation of quality dimensions obtained by time to failure, time to repair, number of failures supported, etc. The statisticalqualifier, unit, and direction, as the core attributes, defines the type of statistical qualifier, the unit for the value dimension and direction (or tendency). The types of statistical qualifiers can be defined as: quality range , maximum value, minimum value, mean, variance, standard deviation, percentile, distribution, etc.
   - QoSDependency, as an extension proposed by [Jureta et al. 2009], expresses the interdependencies between values of quality dimensions.
   - QoSCategory enables grouping QoS characteristics with their priorities to manage a complex class of quality properties sharing similar properties or impacts and facilitate the process of computing service ranking and selection. For instance, grouping enables service modeller to define more abstract quality aspects (e.g., "Performance", which is qualified by stakeholders' perspectives) can comprise *throughput* (the rate of successful service-request completion) and *response time* (the time elapsed between a consumer sending message and receiving response), or "Security" can capture the level and kind of security a service provides and generally include *authentication*, *confidentiality*, *encryption*, and *traceability and auditability*.
   - QoSContext provides supports for the description of quality characteristics.
   - QoSValue instantiates QoS characteristics and determines values for quality dimension.
   - QoSDimensionSlot represents the value of a primitive quality dimension or a reference to another quality value.
   - QoSAggregation, as one of our extensions to the UML QoS meta model, provides methods and supports for end-to-end aggregation of quality values for quality characteristics of a composite service based on variability and composition patterns (See Section 4). The aggRule expresses the rules and operators for quality

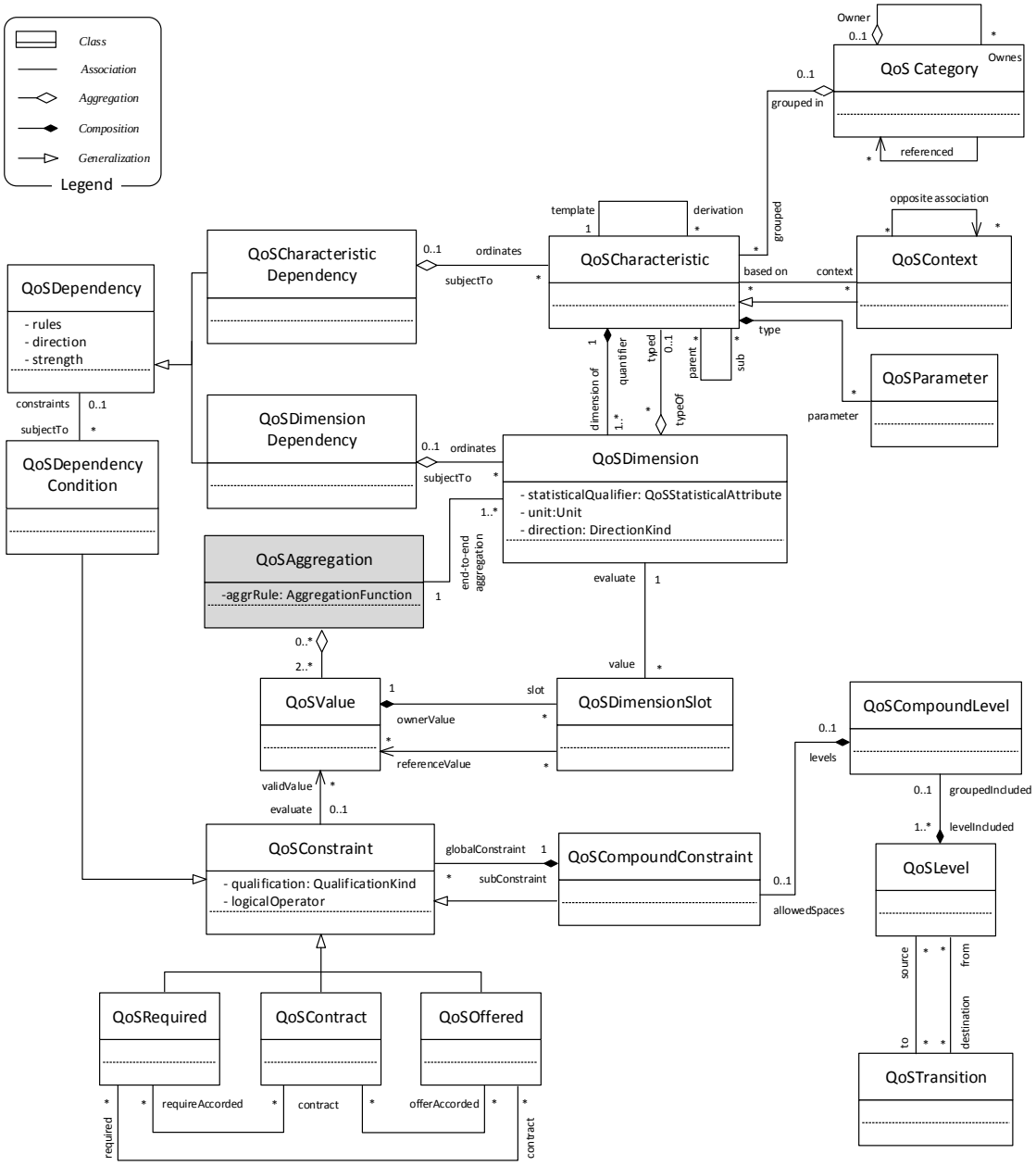


Fig. 3. QoS meta model with proposed extention.

aggregation and computations. Some quality dimensions may follow the same aggregation rule.

(2) QoSConstraint restricts the accepted values of a QoS characteristic, which can be imposed by stakeholders or service provider based on application requirements or architectural decisions. QoScontext expresses the characteristics of quality and functional elements involved in a constraint.
   — QoSCompoundConstraint ensembles constraints and allows to represent global constraints decomposed into a set of sub-constraints.
   — QoSContract composes service and consumer constraints and specifies overall multiple acceptable quality level of service and the requirements required to be achieved.
   — QoSOffer specifies the limits of quality values that service operations can support corresponding to the architectures and implementations designed to supply particular quality.
   — QoSRequire enables service provider or consumer to define quality requirements which are compulsory and systems must fulfil whilst others may be discretional. The required constraints can also be specified either by the service provider, user, or subsytem that needs the consumer to achieve some perquisite level of quality to obtain the quality offered by provider.

(3) QoSLevel determines the working and transition modes under which the service is executed. Quality level defines the quality behavior which is relied on the algorithms, configuration, and infrastructure of the service . Different execution modes may provide distinctive range of quality level for the same service. The details of constructs are available in [SOC 2008].

Figure 4 depicts the submodels of extended UML QoS meta model by preference model allowing to introduce and specify relative importance and conditional constraints over quality characteristics, which are required for decision-making.

— DecisionMaker defines and explicates involved decision-makers (DM) that can be regarded as algorithms, system, service provider, or stakeholder. Decision-maker can stipulate and determine quality relative importance relations in terms of preference or objective function which should be either maximized or minimized for multi-objective optimization and quality-aware service composition and configuration.
— Preference enables decision-makers, i.e., system, service provider, or stakeholder, to express the priorities or relative importance over QoS characteristic, dimension, and category. The preference is often to minimize or maximize an objective function representing preference.
— Precedence allows defining the relative importance relations for QoS model elements to specify the priority between quality pairs. The *rules* determine the order at which characteristic, dimension, and category are further prioritized.
— Weight expresses the strength or degree of importance associated with relative importance relations.
— QoSPrecedenceCondition indicates the constraints when the priorities hold. For instance, the precedence condition specifies priority that should be applied if value over a quality dimension is achieved.
— ObjectiveFunction enables to construct single or multiple objective functions formulating and determining DM's preference. quality-aware optimization may involve more than one objective function which can be constructed by aggregation.
— PreferenceCondition specifies conditional preference on values of quality dimensions to hold.

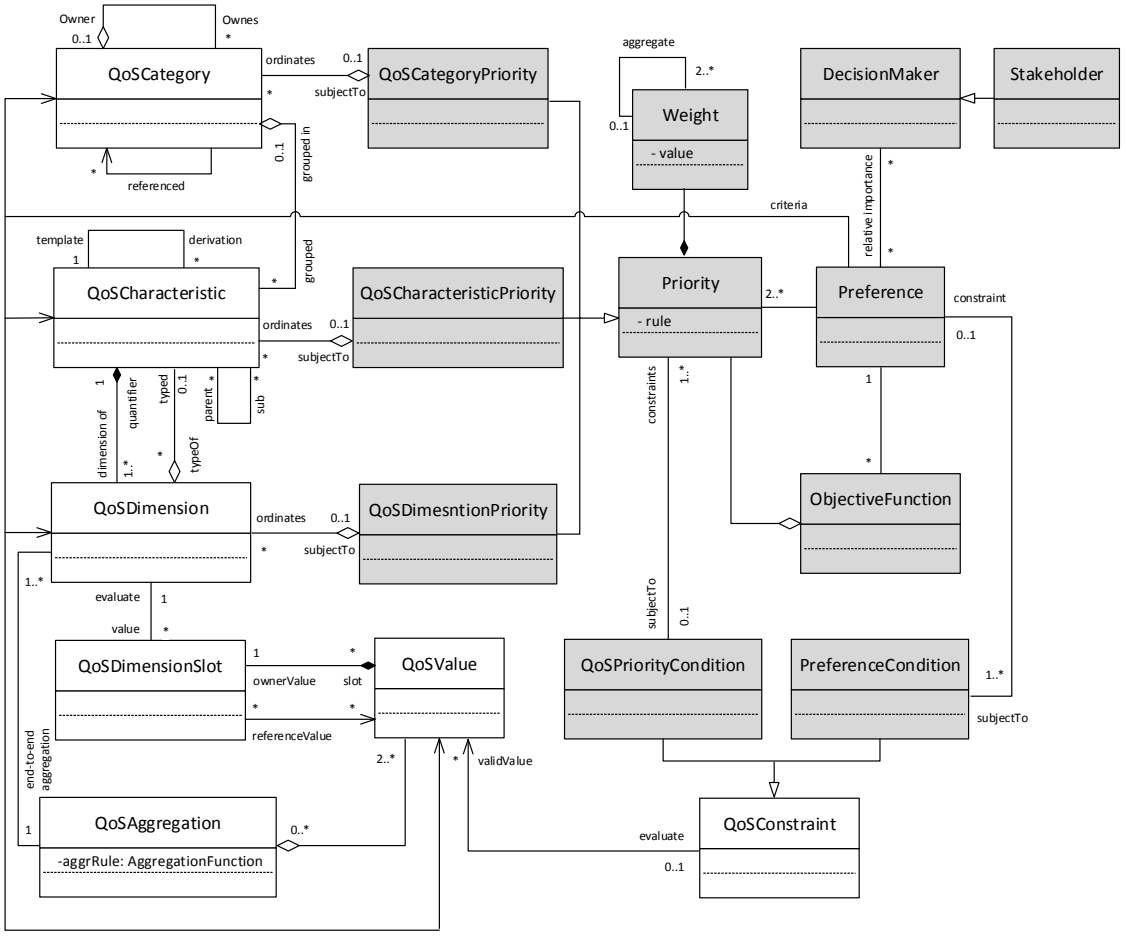


Fig. 4. QoS meta model extended by preference.

4. QUALITY OF SERVICE AGGREGATION AND COMPUTATION FOR PRODUCT LINE ARCHITECTURE

In this section, we describe our proposed quality aggregation model for product line architectures. We will cover the following issues to provide a model for the aggregation and computation of QoS range values in the presence of variability: (a) quality criteria and quality range for SOSPL; (b) combination of variability and composition patterns; and (c) algorithms to aggregate and compute quality range values.

We consider some quantitative QoS characteristics of Web-service, which have been taken into consideration as selection criteria in the research literature [Yu and jay Lin 2005; Zeng et al. 2004; Cardoso et al. 2004; Jaeger et al. 2004]. In this work, we include the following indexed QoS dimensions: 1) *cost*, 2) *response time*, 3) *throughput*, 4) *availability*, which are denoted as $q_{pr}, q_{rt}, q_{tr},$ and q_{av}, respectively. Of course, our approach is not limited to these QoS properties, but these are the only discussed in order to illustrate our approach for quality aggregation and computation.

We also distinguish between two types of quality dimension: *deterministic* and *non-deterministing*. The former indicates that the values of QoS are known or certain when

a service is invoked (e.g., execution price and supported security protocols), whereas the later are unknown or uncertain at invocation time (e.g., response time), therefore, the QoS information should be collected through run-time monitoring. We consider that the QoS information is provided by service broker or middleware.

Based on the underlying implementation of a set of functionally equivalent services, which may be available for each feature, ranges of values of quality dimensions can be further specified and aggregated for each feature. Particularly during the domain engineering life-cycle, determining the implied QoS ranges for individual feature helps service engineers ensure that the product line architecture will fulfill and deliver the upper and lower bounds of the values of the quality requirements requested by the stakeholders. Moreover, quality range computation enables for keeping track of the product line quality ranges even after the specification of the service quality has changed. Mapping models interconnecting feature and business process models enable propagation of quantified quality values of concrete service sets, bounded to activities (abstract services) within in the process model. For example, in Figure 5, sets of candidate services provide different range of quality, denoted as q^R, for each features. The range of the k^{th} quality dimension for feature f can be hierarchically computed. Let us now proceed with some formal definitions as a basis for our work.

Definition 4.1 (***Quality Range***). The quality range values of the ith quality dimension (dimension) is defined as $q_i^{\mathrm{R}} = [q_i^{\mathrm{LB}}, q_i^{\mathrm{UB}}]$, where q_i^{LB} and q_i^{UB} are lower and upper bound values of the quality dimension, respectively.

The above definition shows that each property such as response time or cost can be described by a range of numerical values. This range specifies both lower and upper bounds for that quality dimension. In order to be able to compute such a quality range, appropriate aggregation operators are needed. We consider the following three types of quality aggregation operators for computing the quality ranges of a software product line:

— ***Summation***: The quality range values of the product line is determined by a sum of the QoS range values of the quality attributes of services. An example would be cost;
— ***Multiplication***: The range values of quality attributes are determined by production of the QoS values of the services, for instance, reliability and availability;
— ***Min-Max***: The quality range values of the product line are computed with respect to *critical paths* [Zeng et al. 2004; Dumas et al. 2010] in the business process structure, for instance, response time (i.e., execution duration).

In order to employ the above operators, we consider the following. We assume that for each activity a_n in a business process model BP, there is a bounded set of candidate services, $S_{a_n} = \langle s_{n1}, \ldots, s_{nm} \rangle$, in which all of the candidates provide the same functionality, but with different degrees of quality. The quality of a service s is a vector $Q_s = \langle q_1(s), \ldots, q_k(s) \rangle \in \mathbb{R}$, where the function $q_i(s)$ determines the values of the ith quality dimension.

The quality of each activity a_n is defined as a matrix $[Q_{a_n}]_{i \times j}; 1 \leq i \leq k, 1 \leq j \leq m$, where each row corresponds to a quality dimension q_i, while each column corresponds to a service candidate. Thereby, the range of the ith quality dimension for feature f_n corresponding to activity a_n is obtained by the quality range function $q_i^{\mathrm{R}}(f_n) = [q_i^{\mathrm{LB}}, q_i^{\mathrm{UB}}]$, where $q_i^{\mathrm{LB}} = Q_{an}^{\min}$ and $q_i^{\mathrm{UB}} = Q_{an}^{\max}$.

For example, let us assume that there are five service candidates in S_{a_i} binding to activity a_k, mapped to feature f_k, and that their service cost values (q_{pr}) are given by vector $Q_{a_k}(i,j) = \langle 100, 250, 65, 130, 95 \rangle$. The cost range values of feature f_k could be set

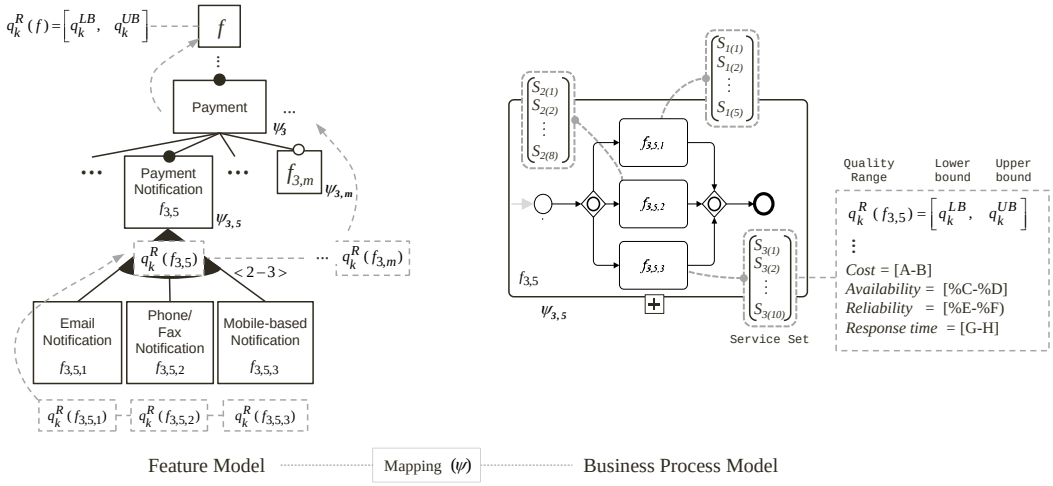


Fig. 5. Non-functional specification and aggregation for evaluating quality range supports by product line architecture

as follows: $q^{\mathrm{R}}_{pr}(f_k) = [65, 250]$, because we are interested in the lower and upper bound values for the quality range.

4.1. Combining Variability and Composition Patterns

In essence, the aggregation model for quality computation in the context of SOSPL depends on: a) structural variability captured by a feature model; and b) behavioral variability captured by a business process model, which describes the composition structure.

Composition patterns[1], which have their roots in workflow management systems[Van Der Aalst et al. 2003], aim at building composition structures that are derived from the requirements in the process modeling phase. In other words, these patterns describe the behavior of features during execution time. Composition patterns are independent of particular composition languages or techniques. and allow coordination of various activities in a process through control elements. They represent the abstract control flow and execution sequence of features within the reference business process model for a service product line. A complete collection of of workflow patterns for the different aspects of process-oriented application development has been presented in [Van Der Aalst et al. 2003]. Similar to [Jaeger et al. 2004], we consider composition patterns addressing behavioral structure of a composition, which are applied for the quality aggregation. These patterns can be grouped into two main groups: a) *sequential patterns* and b) *parallel patterns*. These patterns describe the structure of a process and the execution model for activities (i.e., service).

In a process model, two basic sequential patterns are defined as sequence and arbitrary cycle.

1) **Sequence**: This pattern describes the structure where an activity execute after the completion of another activity in the same process.
2) **Arbitrary cycle:** This pattern, as a special case of sequence pattern, expresses that the execution of one or more activities is repeated for a given number of times. The

[1]We use the terms *workflow* and *composition patterns*, interchangeably.

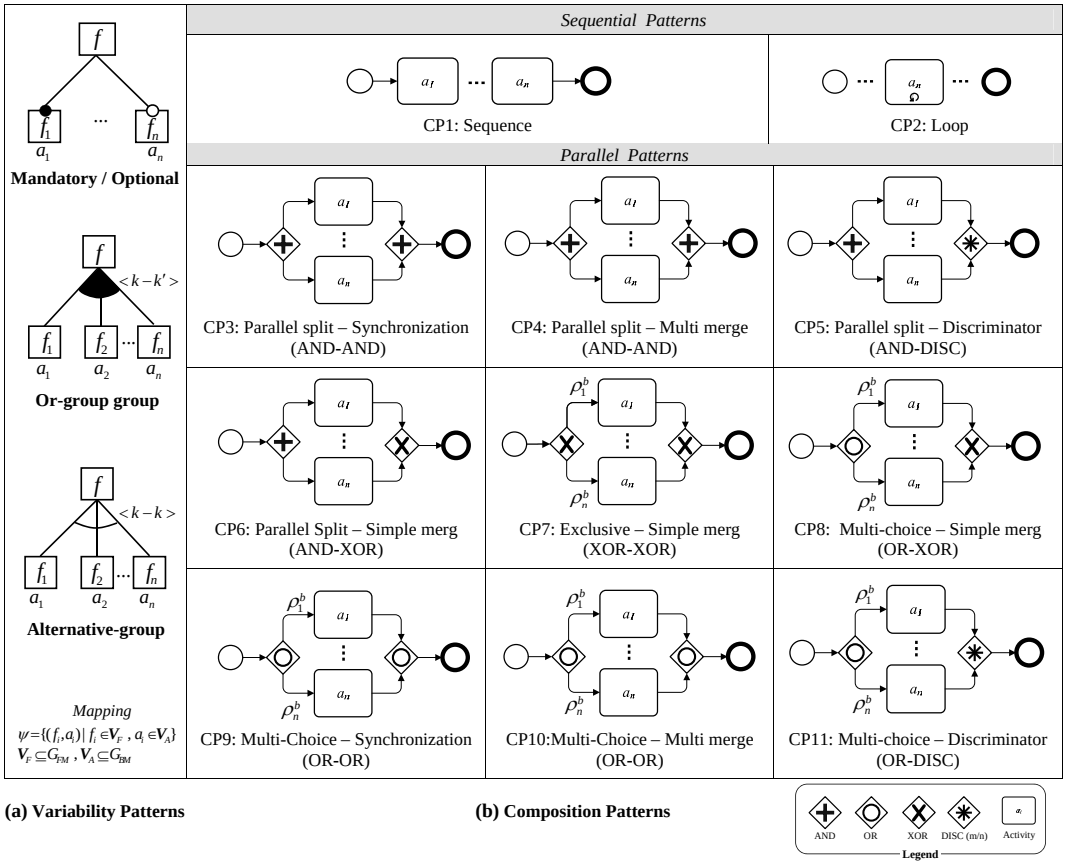


Fig. 6. Variability and composition patterns

number of cycle is determined during the design-time or specified with respect to conditions at run-time.

Parallel patterns describe how the process flow splits into branches for executing the activities and how they merge or converge.

3) **Parallel split:** A parallel split pattern represented by AND-split operation describes the behavioral structure where a single thread of control flow splits into multiple threads which can be executed concurrently in any order.
4) **Multi-choice:** This pattern denoted by OR-split operation describes the structure in which a number of m branches among n ($1 \leq m < n$) in parallel can be selected and executed based on process control data and business rules or specified probability of branches. The ρ_i^b represents the probability of execution of ith branch.
5) **Exclusive/Deferred choice :** An exclusive pattern represented by XOR-split operation describes conditional branching where only one of several branch is selected for the execution based on decision or process control data. Deferred choice resembles the exclusive choice pattern, but in the deferred choice the split is relied on external input, whereas the exclusive choice pattern is based on workflow information (i.e., orchestration data). At design-time, the probability of each branch can

also be determined which specifies the probability that certain branch is chosen to be executed at run-time.

Following general parallel patterns represent how branches model the logic of the convergence of activities that are invoked or executed in a process flow.

6) **Synchronization:** This pattern represented by AND-join operation describes the structure of process where multiple activities in parallel branches converge into one single thread synchronized.
7) **Simple-merge:** The simple merge pattern represented by XOR-join operation describe the structure where multiple branches converge without synchronization and only one of alternative branches has been executed in parallel.
8) **Multi-merge:** This pattern describes the structure where branches converge without synchronization and the activity succeeding the convergence is activated by the completion of every incoming branch.
9) **Discriminator/m-out-of-n-join:** In discriminator pattern, subsequent activity will be executed by the first and only the first branch completed based on the condition and all the rest of branches are disregarded. This pattern also has been known as a partial join [Jaeger et al. 2004; Van Der Aalst et al. 2003]. The discriminator patterns can be generalized where the subsequent activity will be activated only after m-out-of-n activity. The number of m is specified over business process model based on process control data and business rule.

It should be noted that various combinations of parallel split and join cannot semantically and formally provide valid patterns. For example, XOR-split in combination with AND-joint cannot occurs in process, hence it does not generate valid pattern.

Based on comparison of variability and workflow patterns, we first demonstrate how the effects of QoS aggregation for the computation of quality ranges are derived from primitive patterns, and then we describe algorithms to perform the derivation for other complex patterns in the same way. In addition to descriptive definitions of selected workflow patterns, we show the semantics using BPMN. Figure 6 illustrates three main variability patterns (left side), as described in Section 2.3, in combination with 11 composition patterns (CP1-CP11) (right side).

4.2. Aggregation Rules Based on Variability and Composition Patterns

Based on the patterns described above, we define aggregation rules for each QoS property by primarily taking into account the variability patterns which may occur within each composition pattern.

In the following, we present the aggregation rules for four numerical QoS properties: *cost*, *response time*, *availability* and *throughput*. The cost of a feature is the cost which can be associated with the deployment, execution, management, maintenance and monitoring of a service.

The summary of the cost and response time aggregation rules grouped based on mandatory/optional patterns and Or/Alternative variability patterns corresponding to composition patterns are given in Tables I and I, and Table V, respectively. The aggregation rules for availability and throughput are presented in Appendix A.

According to Definition 4.1, the definition of a quality lower bound, q_i^{LB}, for different quality dimensions must indispensably consider the mandatory features for sequential and parallel split patterns (CP1-CP6). In addition to mandatory features, the optional features generally contribute to the upper bound range value, q_i^{UB}. For instance, in the sequential patterns, the cost of particular feature should be determined by the sum of the cost values of each mandatory feature for the lower bound; while the upper bound is resolved by the accumulated cost for mandatory as well as optional features.

Table I. Aggregation rules based on *Mandatory/Optional* patterns for cost and response time.

			QoS property		Cost (q_c)
Mandatory / Optional Patterns	Seq. Patterns	R1	CP1	Sequence	$\left[\sum_{i=1}^{n} q_{pr}^{\mathrm{LB}}(f_i):\forall f_i \in \dot{f} \;,\; \sum_{i=1}^{n} q_{pr}^{\mathrm{UB}}(f_i):\forall f_i \in \dot{f}\vee \mathring{f}\right]$
		R2	CP2	Arbitrary Cycle	$\left[cq_{pr}^{\mathrm{LB}}(f_i): f_i \in \dot{f}\vee \mathring{f} \;,\; cq_{pr}^{\mathrm{UB}}(f_i): f_i \in \dot{f}\vee \mathring{f}\right]$
	Parallel Patterns	R3	CP3 CP4	AND-AND	$\left[\sum_{i=1}^{n} q_{pr}^{\mathrm{LB}}(f_i):\forall f_i \in \dot{f} \;,\; \sum_{i=1}^{n} q_{pr}^{\mathrm{UB}}(f_i):\forall f_i \in \dot{f}\vee \mathring{f}\right]$
		R4	CP5	AND-DISC	
		R5	CP6	AND-XOR	
		R6	CP7	XOR-XOR	$\left[\min\left(q_{pr}^{\mathrm{LB}}(f_i): f_i \in \dot{f}\vee \mathring{f}\right) \;,\; \max\left(q_{pr}^{\mathrm{UB}}(f_i): f_i \in \dot{f}\vee \mathring{f}\right)\right]$
		R7	CP8	OR-XOR	$\left[\min\left(\sum_{f_i \in F_{Sub}} q_{pr}^{\mathrm{LB}}(f_i):\forall F_{Sub} \in F_{C_m^n}\right) \;,\; \max\left(\sum_{f_i \in F_{Sub}} q_{pr}^{\mathrm{UB}}(f_i):\forall F_{Sub} \in F_{C_m^n}\right)\right]$
		R8	CP9 CP10	OR-OR	
		R9	CP11	OR-DISC	

			QoS property		Response Time (q_{rt})
Mandatory / Optional Patterns	Seq. Patterns	R1	CP1	Sequence	$\left[\sum_{i=1}^{n} q_{pr}^{\mathrm{LB}}(f_i):\forall f_i \in \dot{f} \;,\; \sum_{i=1}^{n} q_{pr}^{\mathrm{UB}}(f_i):\forall f_i \in \dot{f}\vee \mathring{f}\right]$
		R2	CP2	Arbitrary Cycle	$\left[cq_{rt}^{\mathrm{LB}}(f_i): f_i \in \dot{f}\vee \mathring{f} \;,\; cq_{rt}^{\mathrm{UB}}(f_i): f_i \in \dot{f}\vee \mathring{f}\right]$
	Parallel Patterns	R3	CP3 CP4	AND-AND	$\left[\max\left(q_{rt}^{\mathrm{LB}}(f_i):\forall f_i \in \dot{f}\right) \;,\; \max\left(q_{rt}^{\mathrm{UB}}(f_i):\forall f_i \in \dot{f}\vee \mathring{f}\right)\right]$
		R4	CP5	AND-DISC	$\left[\min\left(q_{rt}^{\mathrm{LB}}(f_i):\forall f_i \in \dot{f}\right) \;,\; \max\left(q_{rt}^{\mathrm{UB}}(f_i):\forall f_i \in \dot{f}\vee \mathring{f}\right)\right]$
		R5	CP6	AND-XOR	
		R6	CP7	XOR-XOR	$\left[\min\left(q_{rt}^{\mathrm{LB}}(f_i):\forall f_i \in \dot{f}\vee \mathring{f}\right) \;,\; \max\left(q_{rt}^{\mathrm{UB}}(f_i):\forall f_i \in \dot{f}\vee \mathring{f}\right)\right]$
		R7	CP8	OR-XOR	
		R8	CP9 CP10	OR-OR	$\left[\min\left(\max\left(q_{pr}^{\mathrm{LB}}(f_i): f_i \in F_{Sub}, \forall F_{Sub} \in F_{C_m^n}\right)\right) \;,\; \max\left(q_{rt}^{\mathrm{UB}}(f_i):\forall f_i \in \dot{f}\vee \mathring{f}\right)\right]$
		R9	CP11	OR-DISC	

Table II. Aggregation rules based on *Or/Alternative*-feature groups variability patterns for cost and response time.

			QoS Properties		Cost (q_c)	Response Time (q_{rt})
$Or^{(*)1}$ / *Alternative* $^{(**)1}$ *Patterns*	Seq.	R1	CP1	Sequence	$\left[\min\left(\sum_{f_i \in F_{Sub}} q_{pr}^{\mathrm{LB}}(f_i):\forall F_{Sub} \in F_{C_k^n}\right) \;,\; \max\left(\sum_{f_i \in F_{Sub}} q_{pr}^{\mathrm{UB}}(f_i):\forall F_{Sub} \in F_{C_{k'}^n}^{(*)} \middle| F_{C_k^n}^{(**)}\right)\right]$	$\left[\min\left(\sum_{f_i \in F_{Sub}} q_{rt}^{\mathrm{LB}}(f_i):\forall F_{Sub} \in F_{C_k^n}\right) \;,\; \max\left(\sum_{f_i \in F_{Sub}} q_{rt}^{\mathrm{UB}}(f_i):\forall F_{Sub} \in F_{C_{k'}^n}^{(*)} \middle| F_{C_k^n}^{(**)}\right)\right]$
	Parallel Patterns	R3	CP3 CP4	AND-AND	$\left[\min\left(\sum_{f_i \in F_{Sub}} q_{pr}^{\mathrm{LB}}(f_i):\forall F_{Sub} \in F_{C_k^n}\right) \;,\; \max\left(\sum_{f_i \in F_{Sub}} q_{pr}^{\mathrm{UB}}(f_i):\forall F_{Sub} \in F_{C_{k'}^n}^{(*)} \middle| F_{C_k^n}^{(**)}\right)\right]$	$\left[\min\left(\max\left(q_{rt}^{\mathrm{LB}}(f_i):\forall F_{Sub} \in F_{C_k^n}\right)\right) \;,\; \max\left(\max\left(q_{rt}^{\mathrm{UB}}(f_i):\forall F_{Sub} \in F_{C_{k'}^n}^{(*)} \middle| F_{C_k^n}^{(**)}\right)\right)\right]$
		R4	CP5	AND-DISC		
		R5	CP6	AND-XOR		
		R6	CP10	OR-XOR		$\left[\min\left(\min\left(q_{rt}^{\mathrm{LB}}(f_i):\forall F_{Sub} \in F_{C_k^n}\right)\right) \;,\; \max\left(\max\left(q_{rt}^{\mathrm{UB}}(f_i):\forall F_{Sub} \in F_{C_{k'}^n}^{(*)} \middle| F_{C_k^n}^{(**)}\right)\right)\right]$
		R7	CP7 CP8	OR-OR		
		R8	CP9	OR-DISC		$\left[\min\left(\max\left(q_{rt}^{\mathrm{LB}}(f_i):\forall F_{Sub} \in F_{C_k^n}\right)\right) \;,\; \max\left(\max\left(q_{rt}^{\mathrm{UB}}(f_i):\forall F_{Sub} \in F_{C_{k'}^n}^{(*)} \middle| F_{C_k^n}^{(**)}\right)\right)\right]$
		R9	CP11	XOR-XOR		

[1](*) and (**) represent the feature set combinatorial operators which are applied for *Or* and *Alternative* variability patterns, respectively

By adopting a hierarchical approach, described in the next section, the range values (upper and lower bounds) for QoS properties are computed for a combination of variability and composition patterns, based on our formulated aggregation rules. To determine the upper and lower bounds for QoS range values, $q_i^{\mathrm{R}}(f)$, for a parent feature f (see Figure 6), the aggregation rules are applied on the basis of each composition pattern according to the variability patterns.

To compute the quality range values, the aggregation operators are applied to each member set of $F_{Sub} \in F_{C_l^n}$. For instance, for the lower and upper range values of cost (q_{pr}), the summation operator is first applied to each element of F_{Sub}, which results in a new set. For this new set, the min-max operator is then applied.

The mandatory and optional features in the Multi-choice parallel patterns (cf. CP8, CP9 and CP10, and CP11) follow different aggregation rules. To perform quality aggregation, the aggregation model also requires knowing which paths in the business process flow will be chosen at run-time particularly for OR-Splits. However, we consider the worst-case scenarios to determine the lower and upper bounds of quality values and assume that the execution of all possible choices for an OR-Split gateway is equally probable. The business rules defined over business process models (i.e., OR-Split gateway) specify how many paths (m) can be executed at run-time. This results in a feature set $F_{Sub} \in F_{C_m^n}$ where $k \leq m \leq k' \leq n$.

For instance, in our example, assume that two notification features `Email-voicemail` and `Mobile-based notification` are included in an instance of the reference business process. Hence, the decision concerning which notification service should be invoked w.r.t. OR-split semantics is left to run-time.

To address *Or*/*Alternative*-feature groups variability in combination with Multi and Exclusive choice composition patterns (CP7-C11), the aggregation model must consider all possible combinations of features corresponding to the given cardinality $<k..k'>$ over the n features of the feature group specified at design time. Therefore, the resulting feature set (i.e., F_{Sub}) is a subset of $F_{C_k^{k'}}$ and $F_{C_{k'}^n}$ for lower and upper bound quality range values, respectively.

4.3. A Method for QoS Range Aggregation and Computation

Our method is based on analyzing variability and composition patterns expressing the structural and behavioural variabilities of a reference business process.

The QoS range values for features in a feature model are computed by hierarchically aggregating the QoS for variability patterns at the level of each composition pattern. Aggregation is performed by gradually collapsing features into a single feature in the feature model, by employing the notion of a *virtual feature*. This approach enables us to perform the aggregation from both micro and macro perspectives. In other words, the quality range values can be computed for each of the given features from a local view and also for the entire feature model from a global view.

Algorithms 1 and 2 detail the procedure for computing QoS ranges for a feature model. The aggregation and computation of quality range values of particular quality dimension for specific feature starts with algorithm 1, where the feature model is traversed from a given feature node by post-order depth-first traversal, i.e., computing the aggregated QoS ranges from leaves and the right-most nodes up to the root node.

In order to analyze and identify how features at the same level in the feature model are composed in the process model, we decompose the process graph into process components. We employed the Refined Process Structure Tree technique [Polyvyanyy 2012; Vanhatalo et al. 2009] for decomposition and transformation of process model graph and further control flow analysis.

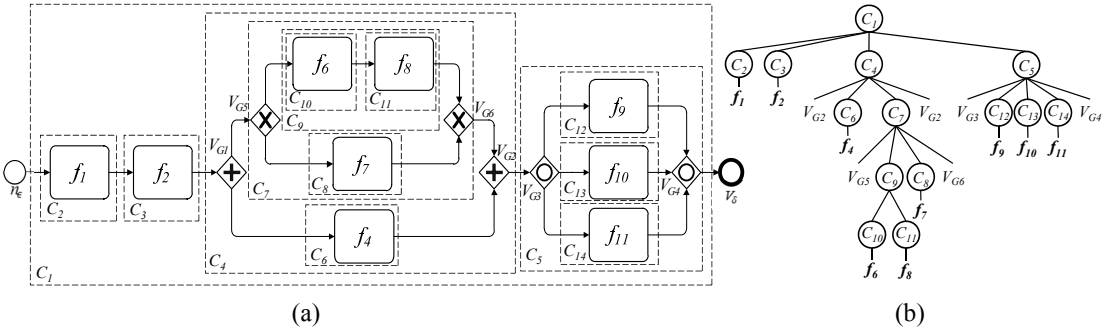


Fig. 7. a) Decomposition of business process model to SESE components. b) Process structure tree.

For each feature, the Process Structure Tree (PST) corresponding to a given feature is subsequently created and parsed. Line 6 in algorithm 1 creates and decomposes the process model graph into a tree of Single-Entry Single-Exit (SESE) components. Each process component is a connected subgraph with a SESE region, which includes all components at the lower level. Each component pertains to one out of four structural classes: a trivial components of a single flow arc; a polygon describing a sequence of components; and a bond presenting a set of components that share two common node. Other components considered as a rigid components. All the SESE components at the same level are disjoint. It is noteworthy that for any process model there is a PST defining unique decomposition [Vanhatalo et al. 2009].

ALGORITHM 1: Aggregate QoS range for feature: AggregateQualityRange(f)

Input: f: given feature of feature model FM
Output: QoS reange- $q^{\mathrm{R}} of feature f$

```
1  begin
       // All direct child features of f;
2      S_f[ ] ⟵ ∀ChildFeatureOf(f) ;
3      if S_f = ∅ then return q^R(f) else
4          for i = 1 to |S_f| do
5              AggregateQualityRange(S_f[i]);
6              PST ⟵ ProcessStructure(S_f[i]);
7              if PST = ∅ then return q^R(f) else q^R(f)= ComputeQualityRange(PST)
8
```

Figure 7(a) depcits the result of decomposition of the exemplified payment process model graph given in Figure 1, where each dotted box represents a SESE component. The resulting process structure tree which defines a hierarchy of components is shown in Figur 7(b). For every trivial SESE component in PST, the control flow analysis is performed, whose details are omitted from algorithm 2 for the sake of brevity; interested readers are referred to [Vanhatalo et al. 2007].

Algorithm 2 operates over a PST, and the aggregation is performed at the level of each processes component. Lines 3 to 24 iteratively aggregate the quality for each child component. For individual process components, features are grouped into virtual features corresponding to the variability patterns. The control flow information is used for identifying the pattern in each of the SESE components. The virtual features, which are denoted as $f_{V_{mo}}, f_{V_{or}}, f_{V_{xor}}$, represent *Mandatory*/*Optional*, and *Or*/*Alternative*-virtual feature group, respectively.

ALGORITHM 2: Compute QoS range values of a process associated to feature f:ComputeQualityRange(C)

Input: C : Process component- node of process structure tree PST
Output: q^{R}: aggregated QoS range of process component

```
1  begin
2    foreach C_i ∈ ChildOf(C) do ComputeQualityRange(C_i) ;
3    forall the f_k ∈ C_i do
       //[⋈]Group features w.r.t. variability patterns and step-wise collapsing
       features by means of virtual features;
4      switch feature f_k.Type do
5        case f_k ∈ f̊ ∨ f•
6          f_Vmo[ ] ⟵ f_k ;
           // Applying aggregation rules for Mandatory/Optional patterns;
7          q^R(f_Vmo) = QoSAggregation(f_Vmo);
8          if ∀f_k ∈ f_Vmo : f_k ∈ f̊ then
9            f_Vmo.Type=f̊
10         else
11           f_Vmo.Type=f•
12       case f_k ∈ Or-group
13         f•_Vor [ ] ⟵ f_k ;
           // Applying aggregation rules for Or pattern;
14         q^R(f•_Vor) = QoSAggregation(f•_Vor) ;
15       case f_k ∈ Alternative-group
16         f•_Vxor [ ] ⟵ f_k ;
           // Applying aggregation rules for Alternative pattern;
17         q^R(f•_Vxor) = QoSAggregation(f•_Vxor);
18
19     f_V[ ] ⟵ f_Vmo, f•_Vor, f•_Vxor;
       // Applying aggregation rules w.r.t. variability patterns;
20     q^R(f_V) = QoSAggregation(f_V);
21     if ∃f_k ∈ f_V : f_i ∈ f• then
22       f_V.Type=f•
23     else
24       f_V.Type=f̊
25   return q^R(f_V)
```

The range values of quality dimension of virtual feature are computed by aggregation function according to the aggregation rules introduced earlier and shown in Tables I,I, and V. It should be noted that aggregation rules are defined for individual quality dimension with respect to the its nature. The class QoSAggregation introduced in QoS model (cf., Figure 3) defines aggregation rules and provides required methods.

In order to comply further with the aggregation rules, the type of virtual features should also be determined. Hence, the type of the corresponding virtual feature is labeled as optional if all the collapsed features are optional, otherwise it is labeled as mandatory (i.e., line 11).

However, it is noted that according to the given semantic descriptions of variability patterns for *Or*/*Alternative*-feature groups, corresponding virtual features $f_{V_{or}}$ and

$f_{V_{xor}}$ are labeled mandatory. The virtual feature f_V includes all of the collapsed virtual features, which are grouped in each basic composition patterns, and its type is determined such that if there is at least one mandatory feature in a grouped feature, the type of the collapsed virtual features is considered mandatory as well (line 21-24).

Example 4.2. To demonstrate the aggregation algorithms described above, we use a simplistic example of the aggregation and computation of QoS range values of cost (q_{pr}) for the payment feature in the feature model shown in Figure 1. It should be noted that different rules are applied for aggregation of range values for other QoS properties,e.g., response time. The following is the step-wise process for computing the QoS range values for the payment feature. To show how each of the transformations is actually performed and how the range values are computed, we refer to the relevant lines of the algorithms that is used besides each step.

$$q^{\mathrm{R}}_{pr}(f) = \left\{ [q^{\mathrm{LB}}_{pr}, q^{\mathrm{UB}}_{pr}] := q^{\mathrm{R}}_{pr}(f_V) | f_V = \textbf{Agg.} \bigcup_{i=1}^{n} f_i \right\} \quad \text{(Alg.1 lines 2-11)}$$

$$q^{\mathrm{R}}_{pr}(\overset{\circ}{f}_1) = [8,20];\ q^{\mathrm{R}}_{pr}(\overset{\bullet}{f}_2) = [17,48];\quad q^{\mathrm{R}}_{pr}(\overset{\circ}{f}_4) = [10,54]; \quad \text{(Alg.1 lines 3)}$$

$$q^{\mathrm{R}}_{pr}(\overset{\bullet}{f}_{V_1}) = \underbrace{\{q^{\mathrm{R}}_{pr}(\overset{\circ}{f}_6), q^{\mathrm{R}}_{pr}(\overset{\bullet}{f}_8)\}}_{\overset{1}{\bowtie}} = \underbrace{\left[q^{\mathrm{LB}}_{pr}(\overset{\bullet}{f}_8)\ , \sum\{q^{\mathrm{UB}}_{pr}(\overset{\bullet}{f}_8), q^{\mathrm{UB}}_{pr}(\overset{\circ}{f}_6)\}\right]}_{\text{Table I-Agg.rule R1}} = [12,47]; \quad \text{(Alg.2 lines 5-8)}$$

$$q^{\mathrm{R}}_{pr}(\overset{\bullet}{f}_3) = \underbrace{\{q^{\mathrm{R}}_{pr}(\overset{\bullet}{f}_{V_1}), q^{\mathrm{R}}_{pr}(\overset{\bullet}{f}_7)\}}_{\overset{2}{\bowtie}} = \underbrace{\left[\min\left(\{q^{\mathrm{LB}}_{pr}(\overset{\bullet}{f}_{V_1}), q^{\mathrm{LB}}_{pr}(\overset{\bullet}{f}_7)\}\right), \max\left(\{q^{\mathrm{UB}}_{pr}(\overset{\bullet}{f}_{V_1}), q^{\mathrm{UB}}_{pr}(\overset{\circ}{f}_7)\}\right)\right]}_{\text{Table I-Agg.rule R6}} \quad \text{(Alg.2 lines 5-8)}$$

$$= [12,51];$$

$$q^{\mathrm{R}}_{pr}(\overset{\bullet}{f}_{V_2}) = \underbrace{\{q^{\mathrm{R}}_{pr}(\overset{\bullet}{f}_3), q^{\mathrm{R}}_{pr}(\overset{\circ}{f}_4)\}}_{\overset{3}{\bowtie}} = \underbrace{\left[q^{\mathrm{LB}}_{pr}(\overset{\bullet}{f}_3)\ , \sum\{q^{\mathrm{UB}}_{pr}(\overset{\bullet}{f}_3), q^{\mathrm{UB}}_{pr}(\overset{\circ}{f}_4)\}\right]}_{\text{Table I-Agg. rule R3}} = [12,105]; \quad \text{(Alg.2 lines 5-8)}$$

$$q^{\mathrm{R}}_{pr}(\overset{\bullet}{f}_5) = \underbrace{\left[\min\left(\sum_{f_i \in F_{Sub}} q^{\mathrm{LB}}_{pr}(f_i) : \forall F_{Sub} \in F_{C^3_2}\right), \max\left(\sum_{f_i \in F_{Sub}} q^{\mathrm{UB}}_{pr}(f_i) : \forall F_{Sub} \in F_{C^3_3}\right)\right]}_{\overset{4}{\bowtie}\ \text{Table V -Agg. rule R7}} \quad \text{(Alg.2 lines 9-11)}$$

$$= \left[\min\left(\sum\{f_9,f_{10}\}, \{f_9,f_{11}\}, \{f_{10},f_{11}\}\right), \max\left(\sum\{f_9,f_{10},f_{11}\}\right)\right] = \left[\min\left(\{18,13,11\}\right), 97\right] = [11,97];$$

$$q^{\mathrm{R}}_{pr}(\overset{\bullet}{f}_{V_3}) = \underbrace{\{q^{\mathrm{R}}_{pr}(\overset{\bullet}{f}_2), q^{\mathrm{R}}_{pr}(\overset{\circ}{f}_5), q^{\mathrm{R}}_{pr}(\overset{\bullet}{f}_{V_2})\}}_{\overset{5}{\bowtie}} \quad \text{(Alg.2 lines 5-8)}$$

$$= \underbrace{\left[\sum\{q^{\mathrm{LB}}_{pr}(\overset{\bullet}{f}_2), q^{\mathrm{LB}}_{pr}(\overset{\bullet}{f}_{V_2})\}\ , \sum\{q^{\mathrm{UB}}_{pr}(\overset{\bullet}{f}_2), q^{\mathrm{UB}}_{pr}(\overset{\circ}{f}_5), q^{\mathrm{UB}}_{pr}(\overset{\bullet}{f}_{V_2})\}\right]}_{\text{Table I-Agg. rule R1}} = [40,250];$$

$$q^{\mathrm{R}}_{pr}(f) = \underbrace{\{q^{\mathrm{R}}_{pr}(\overset{\circ}{f}_1), q^{\mathrm{R}}_{pr}(\overset{\bullet}{f}_{V_3})\}}_{\overset{6}{\bowtie}} = \underbrace{\left[q^{\mathrm{LB}}_{pr}(\overset{\bullet}{f}_{V_3}), \sum\{q^{\mathrm{UB}}_{pr}(\overset{\bullet}{f}_2), q^{\mathrm{UB}}_{pr}(\overset{\circ}{f}_5), q^{\mathrm{UB}}_{pr}(\overset{\bullet}{f}_{V_2})\}\right]}_{\text{Table I-Agg. rule R1}} = [40,270]$$

Figure 8 illustrates the step-wise transformation of the feature model and hierarchical aggregations. In each step, feature model is traversed from a given feature, and variability pattern is determined. The PST corresponding to process model is parsed which is followed by control-flow analysis to detect the composition patterns.

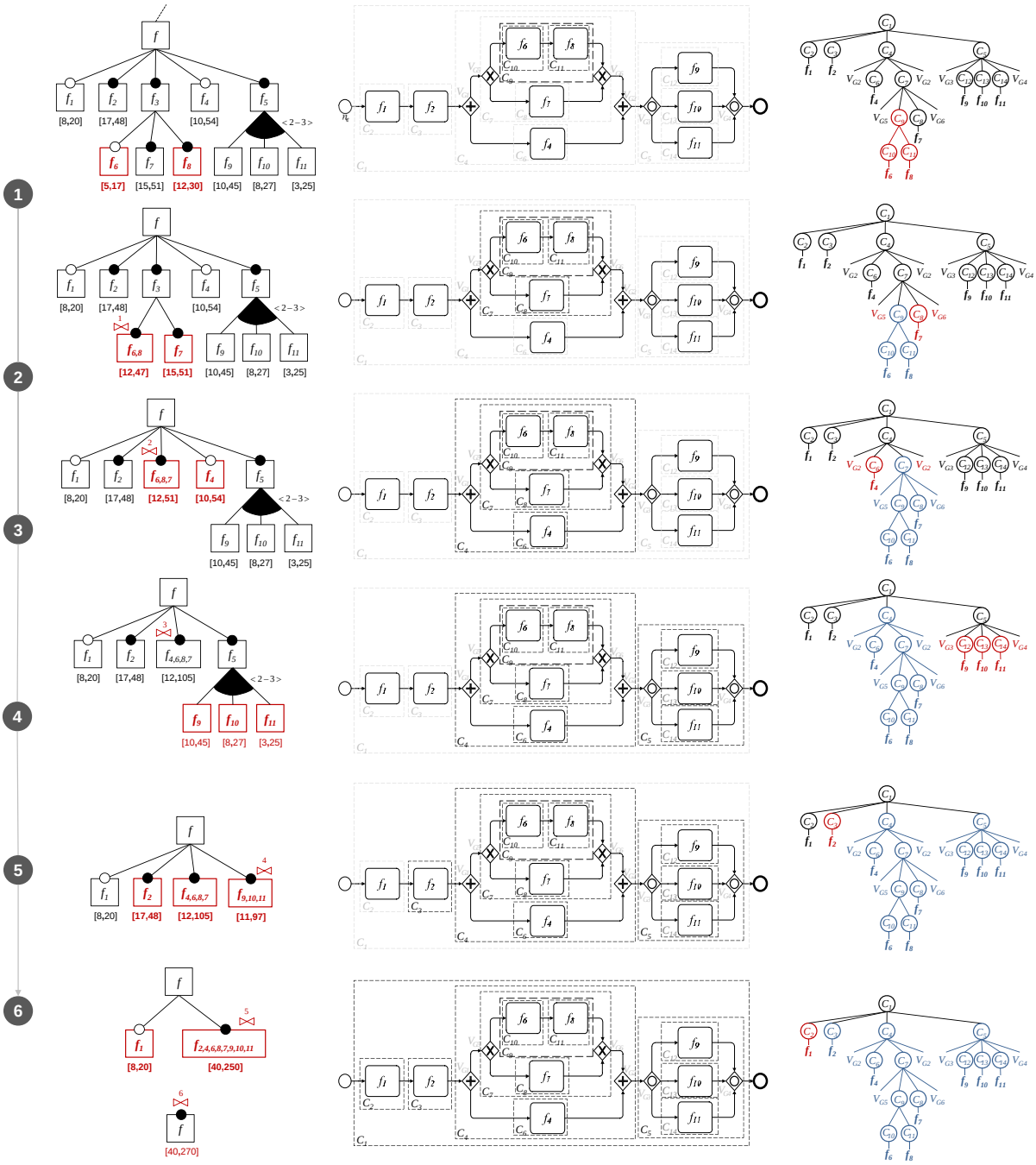


Fig. 8. Step-wise feature model transformation for cost aggregation. a) feature model. b) process model. c) process structure tree.

5. EVALUATION

In this section, we first analyze the computational complexity of the proposed aggregation method, and then we show the performance evaluation which is followed by discussion.

5.1. Complexity Evaluation

The proposed QoS aggregation model includes the following three high-level steps: 1) quality range aggregation of features for feature models; 2) process structure tree construction related to each feature and finding canonical process components based on composition patterns; and 3) aggregation of QoS of process components w.r.t. aggregation rules and their propagation over the feature model.

The size of the state-space in a feature model depends primarily on the size of the given feature model graph $G_{FM}(V, E)$. Backtracking of a feature model produces an ordering of the features in which the parent nodes are placed in post-order of their ancestors. The traversal of a feature model requires $O(|V_{FM}| + |E_{FM}|)$, which has linear time complexity. Given the presence of integrity constraints (includes and excludes relations) in a feature model, the size of the resulting graph will be proportional to the number of constraints. This step could have exponential time complexity for the feature model with large number of constraints. However, we show below that this is usually not the case.

In the second step, the modular decomposition of business processes and the construction of PST is performed in linear-time proportional to the size of the directed graph of the business process (see [McConnell and de Montgolfier 2005]). The third step of the algorithm for computing and aggregating quality range values is achieved by parsing the PST of the business process model $G_{BP}(V, E)$ and control flow analysis via alternative post-order depth-first traversals. This step requires $O((|V_{BP}|+|E_{BP}|).(C+S))$, where C and S denote the execution time of control flow analysis for each process component; and computing quality range values according to both the aggregation rules and the size of candidate service lists for each activity. The time required for grouping features based on variability patterns and capturing quality values does not add any computational complexity and can hence be ignored.

As a result, given that the worst-case time complexity of the first step can be in some cases exponential, the time complexity of the entire algorithm can be exponential in the worst-case. However, it is important to note that the aggregation algorithm has a linear time complexity when the number of integrity constraints is smaller than the number of features in a feature model. The number of integrity constraints to the number of features ratio is approximately 18%. This is usually the case based on our analysis of feature models at S.P.L.O.T [2] [Mendonca et al. 2009], hosting a repository of feature models that are publicly available and distributed by the software product line community.

5.2. Performance Evaluation

5.2.1. Experimental Framework and Settings. To evaluate the performance and scalability of QoS aggregation algorithms described above, we conducted a set of experiments based on extensive simulations including different scenarios and settings by real and synthetic QoS data sets. We employed simulation modeling technique by following guidelines similar to those proposed in [Kellner et al. 1999]. Simulation is a widely-used research method to study and analyse complex scenarios and to gain insights

[2] http://splot-research.org/

into performance and scalability for large-size problem instances. Moreover, it helps to evaluate the generalizability of the results. For the evaluation of complex configuration scenarios, we need the ability to construct models with different specifications ranging from small to large-sized models with respect to diverse structural and behavioral characteristics. For this purpose, we developed a simulation framework allowing automatic generation and steering of testbeds of large and complex structural and behavioral models. We calculate the average of the required computation cost for each setting. The experiments were performed on an Intel Xeon Dual CPU 2.8 GHz processor with 8 GB of memory with Windows 7 and Java Virtual Machine 1.6.

5.2.2. Dataset Description. In our evaluation, we experimented with real QoS datasets [Zheng et al. 2010] and [Al-Masri and Mahmoud 2008] as the baseline to generate different distributions and QoS values. These datasets publicly available which comprise measurements of QoS attributes of real-world Web services collected from public sources on the Web. We calculated the computation cost for aggregation of five quality dimensions cost, response time, availability, throughput and reliability. QoS datasets[3] [Zheng et al. 2010] includes the quality evaluation results from 339 users on 5,825 Web services for response time and throughput. QoS dataset[4] [Al-Masri and Mahmoud 2008] comprises 2,507 Web services results of evaluation of one user with the measurements of nine QoS attributes: response time, availability, throughput, likelihood of success, reliability compliance, best practices, latency, and documentation.

5.2.3. Evaluation Results. We consider the execution time of algorithms as the performance criteria during the quality aggregation and computation, and take into account the scalability and performance affected by the size of references process models. We generated reference process model instances with three different sizes in terms of numbers of activities (n_a) and random topology: n_a^1=500, n_a^2=600, and n_a^3= 1000 activities. An empirical study provides evidence that process models in practice and industrial collections include on average 20 activities [Mendling 2009]. However, we considered the SAP reference model as our baseline which is considered to be a representative of a large-sized reference model. SAP reference model is a collection containing about 600 process models. In order to achieve a high internal validity, we generated a very large process model with size of 1000 activities.

All the generated models are constituted of equal ratio of random composition patterns. Furthermore, for individual process model, a feature model and corresponding mapping model is generated with equal ratios of variability patterns.

To evaluate the scalability of our approach, we also examined the impact of different process model sizes (i.e., n_a^1, n_a^2, and n_a^3) in terms of number of Web-service alternatives per activities (n_s) varying from [1,2] to [64,128]. Figure 9 reports the average computation cost of quality aggregation and impacts of process model size varying in number of activities and service candidates per activity. For the formation of fairly large process model (i.e., 1000 activities with a range of 64 to 128 Web service candidates per activity), it can be observed the algorithm performs quality aggregation in reasonable time, i.e., the average computation cost is 26.38 seconds. Nevertheless, such large sets of Web-service candidates and settings are not encountered normally in industrial practice and have been considered to be worst-case scenario in our evaluation from theoretical point of view. Table III presents the detail of statistics.

In addition, we performed linear regressions to estimate the computation cost with respect to process model size. In Figure 9, regression linear lines present the relationship between independent variable (x) as computation cost, and dependent (y)

[3]http://www.wsdream.net/dataset.html
[4]http://www.uoguelph.ca/~qmahmoud/qws/

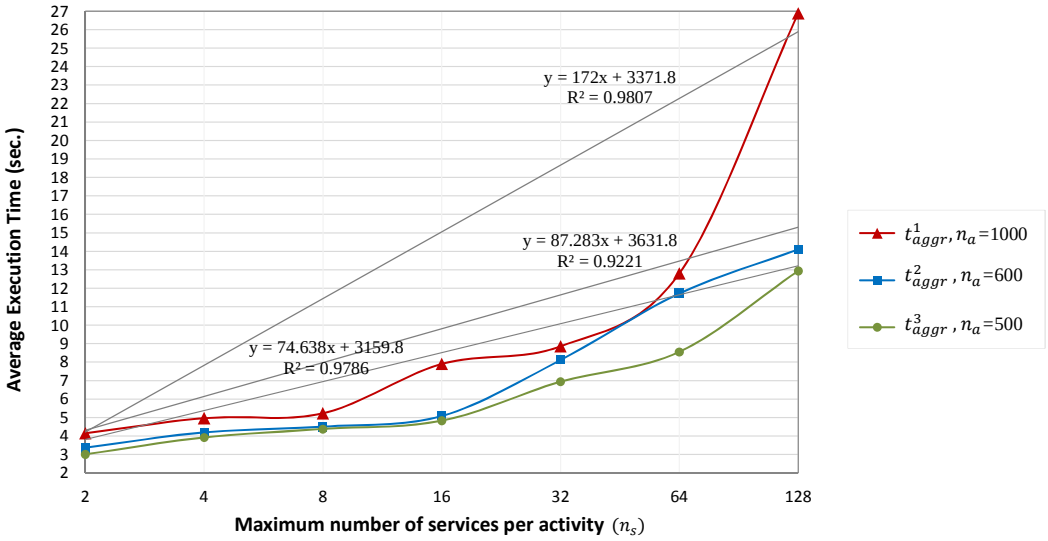


Fig. 9. Computation cost for QoS aggregation for different process model size.

variables as process model size. We use the Pearson correlation coefficient (R) as a measure of precision to determine the quality of prediction and measure the reliability of the simulation forecasts. The Pearsons R measures and indicates the strength and direction of linear association, and the coefficient of determination (R^2) expresses the outcome of the fraction of variation. The range of R^2 lies between zero and one and the value close to one, is considered an indicator to describe better prediction. Straight lines with high coefficient of determination depict linear trend in given graphs. The results indicate the linear relationship between computation cost and size of process model in terms of number of service candidates per each activity.

Based on experiments, for large-size models, the trend line reveals a moderately strong linear relationships between computation cost and the sizes of process model and feature model.

Table III. Statistics of computation cost for different process model size.

n_s \ n_a	500				600				1000			
	Min (sec.)	Max (sec.)	Mean (sec.)	Std. (sec.)	Min (sec.)	Max (sec.)	Mean (sec.)	Std. (sec.)	Min (sec.)	Max (sec.)	Mean (sec.)	Std. (sec.)
[1,2]	2.22	2.96	2.51	0.12	2.45	3.25	2.86	0.16	3.12	4.09	3.64	0.22
[3,4]	3.17	4.01	3.42	0.18	3.42	4.31	3.70	0.18	4.06	5.77	4.47	0.25
[5,8]	3.50	4.57	3.88	0.20	3.70	4.51	4.00	0.18	4.34	5.30	4.73	0.20
[9,16]	3.95	4.85	4.33	0.20	4.17	5.13	4.58	0.25	6.93	8.24	7.40	0.24
[17,32]	6.07	8.58	6.44	0.31	7.04	8.61	7.62	0.27	7.61	10.31	8.36	0.42
[33,64]	7.49	8.97	8.06	0.26	9.42	12.29	11.22	0.39	11.30	14.01	12.31	0.54
[65,128]	11.67	13.54	12.44	0.41	12.42	15.55	13.61	0.59	20.12	32.79	26.38	1.94

As mentioned earlier, we have made some assumptions regarding the topology of the business process graph, i.e., the proposed aggregation method is performed over well-structured business process models. Well-structured business processes have a number of desirable properties, which result in less sophisticated verification mechanisms [Kiepuszewski et al. 2000]. However, such an assumption requires the transfor-

mation of any ad hoc business process graph into a structural business process model. Substantial amount of work for the transformation of unstructured and arbitrary business process models to structured models already exists [Dumas et al. 2010; Ouyang et al. 2009; Vanhatalo et al. 2009; Kiepuszewski et al. 2000]. Therefore, our proposed approach can be further adapted and applied to both structured and unstructured business process models (through transformation). The limitation of the presented aggregation model is that it has not considered the integrity constraints and dependencies between features to deliver a more precise evaluation of quality range values. This is left for future work.

Revisiting our original formulated challenge, the goal of QoS range aggregation is to ensure that the required quality levels are achieved for SOSPL architecture for each product in the family. The proposed quality aggregation model considers variability patterns from both structural and behavioral perspectives. The presented approach is a step towards achieving quality-aware product line configuration. Even in this early stage of the development, this approach supports quality-aware staged configuration. It can be used to assure that every stage yields a subset of products whose quality ranges satisfy the desired QoS ranges.

6. RELATED WORK

Our work is related to a number of different areas. In the following sections, we first overview the approaches focusing on modeling non-functional requirements and quality properties in SPLs. Then, we describe approaches addressing quality evaluation for SPL architectures. In the last section, we summarize approaches that aim to address QoS aggregation and computation for SOA.

6.1. Modeling Quality Characteristics in SPLs

A literature survey on approaches addressing non-functional requirements is presented in [Chung and Prado Leite 2009]. A number of research efforts have developed different modeling approaches and evaluation methods for quality characteristics in the area of product line engineering.

There are several proposals about inclusion of quality information in feature models to support quality-driven feature selection and product line configuration [D. Benavides and Cortés 2010]. Benavides et al. [2005] proposed an extension to feature model to accommodate non-functional properties (quality attributes) in features and facilitate further reasoning and automatic selection of features. The relations among features and feature attributes are transformed into a constraint satisfaction problem (CSP) for quality-aware product configuration. Thurimella et al. [2008] proposed an extended feature model by augmenting selection criteria, as product quality attributes for each variant feature. Their proposed method assesses each variable feature based on its related criteria in qualitative values. They leveraged orthogonal variability modeling (OVM) [Pohl et al. 2005] to separate concern between variability and system modeling and extended rationale model–questions, options and criteria (QOC) [MacLean et al. 1991] to determine rationale behind decisions for requirement engineering.

Goal-oriented approaches is also utilized to model quality requirements. Jarzabek et al. [2006] proposed an approach based on soft-goal modeling technique and a framework to model quality attributes by a goal model linked to a feature model. Their framework integrates goal and feature models by introducing the notion of feature-softgoal interdependency graph (F-SIG) which enables to express the dependency among quality attributes and features. Nevertheless, their approach is limited to the feature model, and they did not take into account the variability in system behaviour. In similar work, Yu et al. [2008] employed goal models to express and capture the

stakeholder's goals or intentions expressed in terms of quality attributes associated to features.

For variability modeling and quality evaluation, other methods also have been proposed that are not based on feature models. Deelstra et al. [2009] developed a method and framework, named COVAMOF software variability assessment method (COSVAM), to support modeling variation points and dependencies in a product line. They introduced the notion of dependency to represent system properties, such as quality attributes, and specify how the selection of individual variant features influences the system properties. In another work, Niemelä et al. [2007] introduced a method, called Quality Requirements of a Software Family (QRF), to support modeling and defining quality requirements, and mapping requirement. However, these approaches did not address how to explicitly model quality and evaluate quality variability.

6.2. Evaluating Quality Characteristics in SPLs

Bartholdt et al. [2009] studied the relevance of quality characteristics to the choice of specific features for non-trivial software product lines. They proposed the integration of quantitative and qualitative quality attributes with features by incorporating quality modeling based on UML profile and feature modeling. Their method only includes aggregated quality evaluation capabilities at modeling level, where the overall quality of each product is computed using predefined aggregation functions. Authors discussed that the quality evaluation requires to devise quality aggregation methods and computational algorithms which consider both structural variability and behavioural models. They also argued that holistic modeling and evaluation of quality of SPLs comprising large set of variations face significant challenges raised by these requirements.

Etxeberria et al. [2008] discussed on the importance of quality analysis of product lines and argued how it can be more scalable through architecture evaluation. To reduce the evaluation efforts, their approach used an extended feature model to identify the variability influencing on quality. In their approach, feature model is extended with additional quality feature tree that characterizes quality attributes and utility function for each feature. The impact of individual features on a particular quality attribute is quantified and measured by evaluating product line architecture through the execution of final configured product. In particular, for performance analysis of architecture, they used execution graph, which is derived from codes, and defined mathematical formula to quantify performance value. Quality formulas include variable quality properties reflecting variable part of the product line architecture. However, it is not address how to compute such those quality values.

Zhang et al. [2003] proposed an approach to predict and evaluate quality variants of an SPL architecture using Bayesian Belief Networks (BBN). In their approach, feature model is used to model functional variants, and BBN is employed to represent the design decisions and quality attributes, and predict the impacts of different configurations on product quality. The BBN model includes probability numbers to reflect domain expert's belief in how much a given configuration influences a quality attribute.

There is limited work in software quality engineering that aims at investigating how to evaluate specific quality characteristics of SPL architectures. For instance, Immonen et al. [2006] proposed method for availability and reliability predictions. Olumofin et al. [2007] presented a method to examine the selection of variant features on availability, reliability, performance, and modifiability of a product line architecture. They proposed a scenario-based method, as Holistic Product Line Architecture Assessment (HoPLAA), which is an adaptation of the Architecture Trade-off Analysis Method (ATAM) [Kazman et al. 2000] for the area of software product lines. For the quality evaluation of SPL architectures, the ATAM is extended with the qualitative

analytical treatment of variation points and the context-dependent generation, and prioritization of quality attributes scenarios.

Sincero et al. [2010] proposed a feedback-based approach to predict product's non-functional properties based on the knowledge base including the quality information and the measurements of product which is already configured. They aimed at providing qualitative information about how each feature and combination of features impact on a certain non-functional property by collecting quality information through the generation and testing products and processing the results. In recent work, Siegmund et al. [2013; 2012] presented a method for the prediction of non-functional properties in software product line based on product's feature selection. In their approach, a set of configuration paths for products are generated that are different in the single feature. Then, the variation of quality properties are measured to identify and approximate the influence of particular feature.

The majority of conducted studies outline and highlight the need for approaches to address the complexity of quality evaluation of SPL because of the involvement of large number of quality attributes scenarios. Most existing works only focus on the evaluation of SPL architecture by assessing and predicting the relative impacts of feature selection on quality in a product-line. However, these attempts did not provide solutions, nor address how quality variability and quality ranges can be obtained.

6.3. QoS Aggregation and Computation

There are several contributions and previous studies addressing QoS aggregation in terms of different process model structures for composite services. The works of Cardoso et al. [2004] and Jaeger et al. [2004] are the seminal works in the literature and basis of our work, which address the aggregation and estimation of QoS values for Web services composition in well-structured process models. Their approaches are based on (some) workflow patterns in the work by van der Aalst et al. [2003]. Cardoso et al. [2004] proposed Stochastic Workflow Reduction (SWR) to compute and estimate the entire workflow QoS values. The SWR algorithm iteratively applies a set of reduction rules for some sequential and parallel patterns over a given structured process graph. Their proposed algorithm for aggregation makes it possible to predict the QoS performance of the entire process by repeatedly performing substitution until the whole process is transformed into one composite service node. Jaeger et al. [2004] proposed a QoS aggregation method which is the most similar to ours. In their approach, quality aggregation of composite Web services is performed by considering workflow patterns and computing upper and lower bounds for QoS values. Authors represent a process model as a graph which is collapsed step-by-step by applying composition patterns. Hwang et al. [2007] proposed a probability-based method where a composite service is represented by a process structure, which is recursively parsed and analyzed to aggregate quality attributes. Mukherjee et al. [2008] focused on addressing the QoS computation in BPEL processes by performing aggregation on an activity graph. In a more recent work, Yong et al. [2012] presented an aggregation method that employs RPST and supports unstructured process models.

Most of the above studies are related to our proposal. However, existing solutions do not consider the constituent *structural variability* of process models and do not address modeling and managing variation points, which may occur within such an architecture and can significantly impact the proper QoS aggregation. To the best of our knowledge, this is the first work that takes both structural and behavioral variability into account to evaluate QoS dimensions in the context of SOSPL.

7. CONCLUSION

In this paper, we proposed a QoS model and provided a systematic approach for QoS aggregation to address the evaluation of quality ranges supported by SOSPL(s), which further help for quality-aware service-product derivation and decision-making and optimization. As the main contribution, we identified a set of variability patterns which may occur within composition patterns and proposed new aggregation rules and a method for QoS range computation for holistic evaluation and prediction of quality ranges of a product-line of services based on configurable process model.

By integrating quality modeling into SOSPL, we incorporate the concepts and constructs to define quality properties from different decision-makers' perspectives. The proposed QoS model provides expressive and extensible submodels to specify relations, dependencies, and relative importance of quality characteristics which are indispensable for quality-driven service configuration and optimization.

The quality assessment of product line is more complex and sophisticated than single service product because of the existence of variation points in terms of functionality and quality properties. Alternative design decisions and variant architectural styles in an SOSPL can be applied to configure service products with the same functionality but different quality levels. Our approach provide a generic evaluation model, the foundation and starting point for development of methods for quality evaluation and assurance, and architectural trade-off analysis of SOSPLs.

We presented a feature-oriented approach to analyze variability and composition patterns to aggregate QoS ranges for an SOSPL. We proposed extended feature model to explicate the range of functional variants accounted for in the product line of services and consider the impacts of variable features on quality aspects. Extended feature model is used to guide feature selection to derive or configure specific service-products. We utilized an holistic approach that enables automatic computation of QoS ranges values for a configurable process models and product line architecture. The results of our evaluation demonstrate the performance and scalability of approach for very large process models. This work can be considered to support variability of enterprise services, and facilitate the management and configuration of multi-tenant cloud applications.

There are several aspects of this work that could be improved and expanded , and further investigation can help to make the proposed quality evaluation framework more practical. There is a need to study the implications when non-hierarchical dependency relationships exist among features in terms of integrity constraints and complex relations. We identified that the integrity constraints may not be completely independents. Therefore, the feature interaction, selection or presence of a particular feature in the final service product may impose intricate chain of constraints which influence on range of quality characteristics. In consequence, measuring and determining exact values of feature's impacts on a quality characteristic is usually not possible. As many efforts have been dedicated to study feature interaction and dependency analysis in software product lines research which can also be considred [Siegmund et al. 2013; Siegmund et al. 2012; Mendonça et al. 2008; Lee and Kang 2004]. For instance, heuristic-based approaches can be utilized to detect feature interactions [Siegmund et al. 2012].

Another direction in which our work can be extended is to leverage probabilistic modelling approaches to provide probabilistic analysis and assessment of quality properties that are expressed in a probabilistic form and enable reasoning under uncertainty. Probabilistic modelling is widely used in the field of performance evaluation, i.e., Bayesian Network, Discrete-Time, Markov chains (DTMCs), Markov decision processes (MDPs) and continuous time Markov chains (CTMCs).

Our future research direction includes quality-aware configuration of service-oriented product lines and optimization. Our current efforts focus on developing a framework for configurable process models and methods for the optimization, preference-based ranking and feature selection based on stakeholders' preferences about functional and QoS requirements.

A. APPENDIX

Table IV. Aggregation rules based on *Mandatory/Optional* patterns for availability and throughput.

			QoS property		Availability (q_{av})
Mandatory / Optional Patterns	Seq. Patterns	R1	CP1	Sequence	$\left[\prod_{i=1}^{n} q_{av}^{\mathrm{LB}}(f_i):\forall f_i \in \dot{f} \vee \mathring{f} \;,\; \prod_{i=1}^{n} q_{av}^{\mathrm{UB}}(f_i):\forall f_i \in \dot{f}\right]$
		R2	CP2	Arbitrary Cycle	$\left[q_{av}^{\mathrm{LB}}(f_i)^{C}:\forall f_i \in \dot{f} \vee \mathring{f} \;,\; q_{av}^{\mathrm{UB}}(f_i)^{C}:\forall f_i \in \dot{f} \vee \mathring{f}\right]$
	Parallel Patterns	R3	CP3 CP4	AND-AND	$\left[\prod_{i=1}^{n} q_{av}^{\mathrm{LB}}(f_i):\forall f_i \in \dot{f} \;,\; \prod_{i=1}^{n} q_{av}^{\mathrm{UB}}(f_i):\forall f_i \in \dot{f} \vee \mathring{f}\right]$
		R4	CP5	AND-DISC	
		R5	CP6	AND-XOR	
		R6	CP7	XOR-XOR	$\left[\min\left(q_{av}^{\mathrm{LB}}(f_i):\forall f_i \in \dot{f} \vee \mathring{f}\right) \;,\; \max\left(q_{av}^{\mathrm{UB}}(f_i):\forall f_i \in \dot{f} \vee \mathring{f}\right)\right]$
		R7	CP8	OR-XOR	$\left[\min\left(\prod_{f_i \in F_{Sub}} q_{av}^{\mathrm{LB}}(f_i):\forall F_{Sub} \in F_{C_m^n}\right) \;,\; \max\left(\prod_{f_i \in F_{Sub}} q_{av}^{\mathrm{UB}}(f_i):\forall F_{Sub} \in F_{C_m^n}\right)\right]$
		R8	CP9 CP10	OR-OR	
		R9	CP11	OR-DISC	

			QoS property		Throughput (q_{tp})
Mandatory / Optional Patterns	Seq. Patterns	R1	CP1	Sequence	$\left[\min\left(q_{tp}^{\mathrm{LB}}(f_i):\forall f_i \in \dot{f} \vee \mathring{f}\right) \;,\; \max\left(q_{pr}^{\mathrm{UB}}(f_i):\forall f_i \in \dot{f} \vee \mathring{f}\right)\right]$
		R2	CP2	Arbitrary Cycle	$\left[q_{tp}^{\mathrm{LB}}(f_i):f_i \in \dot{f}_i \vee \mathring{f}_i \;,\; q_{tp}^{\mathrm{UB}}(f_i):f_i \in \dot{f}_i \vee \mathring{f}_i\right]$
	Parallel Patterns	R3	CP3 CP4	AND-AND	$\left[\min\left(q_{tp}^{\mathrm{LB}}(f_i):\forall f_i \in \dot{f} \vee \mathring{f}\right) \;,\; \min\left(q_{pr}^{\mathrm{UB}}(f_i):\forall f_i \in \dot{f} \vee \mathring{f}\right)\right]$
		R4	CP5	AND-DISC	
		R5	CP6	AND-XOR	$\left[\min\left(q_{tp}^{\mathrm{LB}}(f_i):\forall f_i \in \dot{f} \vee \mathring{f}\right) \;,\; \max\left(q_{tp}^{\mathrm{UB}}(f_i):\forall f_i \in \dot{f} \vee \mathring{f}\right)\right]$
		R6	CP7	XOR-XOR	
		R7	CP8	OR-XOR	
		R8	CP9 CP10	OR-OR	$\left[\min\left(q_{tp}^{\mathrm{LB}}(f_i):\forall f_i \in \dot{f} \vee \mathring{f}\right) \;,\; \max\left(\min\left(q_{tp}^{\mathrm{UB}}(f_i):f_i \in F_{Sub},\forall F_{Sub} \in F_{C_m^n}\right)\right)\right]$
		R9	CP11	OR-DISC	

REFERENCES

2008. *UML Profile for Modeling QoS and FT Characteristics and Mechanisms v1.1*. Technical Report. Object Management Group. http://www.omg.org/technology/documents/formal/QoS_FT.htm

Wil M. P. van der Aalst, Alexander Hirnschall, and H. M. W. (Eric) Verbeek. 2002. An Alternative Way to Analyze Workflow Graphs. In *Proc. of the 14th Int. Conference on Advanced Information Systems Engineering (CAiSE '02)*. Springer-Verlag, London, UK, UK, 535–552.

Eyhab Al-Masri and Qusay H. Mahmoud. 2008. Investigating web services on the world wide web. In *Proceedings of the 17th international conference on World Wide Web (WWW '08)*. ACM, New York, NY, USA, 795–804.

Table V. Aggregation rules based on *Or/Alternative*-feature groups variability patterns for availability and throughput.

QoS Properties					Availability (q_{av})	Throughput (q_{tp})
Or/ Alternative Patterns	Seq.	R1	CP1	Sequence	$\left[\min\left(\prod_{fi\in FSub} {}^{LB}q_{av}(f_i):\forall F_{Sub}\in F_{C_k^n}\right), \max\left(\prod_{fi\in FSub} {}^{LB}q_{av}(f_i):\forall F_{Sub}\in {}^{(*)}F_{C_{k'}^n}\,|\,{}^{(**)}F_{C_k^n}\right)\right]$	$\left[\min\left(\min\left({}^{LB}q_{tp}(f_i):\forall F_{Sub}\in F_{C_k^n}\right)\right), \max\left(\max\left({}^{UB}q_{tp}(f_i):\forall F_{Sub}\in {}^{(*)}F_{C_{k'}^n}\,|\,{}^{(**)}F_{C_k^n}\right)\right)\right]$
	Parallel Patterns	R3	CP3 CP4	AND-AND		$\left[\min\left(\min\left({}^{LB}q_{tp}(f_i):\forall F_{Sub}\in F_{C_k^n}\right)\right), \min\left(\min\left({}^{UB}q_{tp}(f_i):\forall F_{Sub}\in {}^{(*)}F_{C_{k'}^n}\,|\,{}^{(**)}F_{C_k^n}\right)\right)\right]$
		R4	CP5	AND-DISC		
		R5	CP6	AND-XOR		
		R6	CP10	OR-XOR		
		R7	CP7 CP8	OR-OR		$\left[\min\left(\min\left({}^{LB}q_{tp}(f_i):\forall F_{Sub}\in F_{C_k^n}\right)\right), \max\left(\max\left({}^{UB}q_{tp}(f_i):\forall F_{Sub}\in {}^{(*)}F_{C_{k'}^n}\,|\,{}^{(**)}F_{C_k^n}\right)\right)\right]$
		R8	CP9	OR-DISC		
		R9	CP11	XOR-XOR		

Aldeida Aleti, Barbora Buhnova, Lars Grunske, Anne Koziolek, and Indika Meedeniya. 2013. Software Architecture Optimization Methods: A Systematic Literature Review. *IEEE Trans. Softw. Eng.* 39, 5 (May 2013), 658–683.

Mohammad Alrifai, Thomas Risse, and Wolfgang Nejdl. 2012. A hybrid approach for efficient Web service composition with end-to-end QoS constraints. *ACM Trans. Web* 6, 2, Article 7 (June 2012), 31 pages.

Alain Andrieux, Karl Czajkowski, Asit Dan, Kate Keahey, Heiko Ludwig, Toshiyuki Nakata, Jim Pruyne, John Rofrano, Steve Tuecke, and Ming Xu. 2005. *Web Services Agreement Specification(WS-Agreement)*. Technical Report. Global Grid Forum, Grid Resource Allocation Agreement Protocol (GRAAP) WG.

D. Ardagna and B. Pernici. 2007. Adaptive service composition in flexible processes. *IEEE Transactions on Software Engineering* (2007), 369–384.

Joerg Bartholdt, Marcel Medak, and Roy Oberhauser. 2009. Integrating Quality Modeling with Feature Modeling in Software Product Lines. In *Proceedings of the 2009 Fourth International Conference on Software Engineering Advances (ICSEA '09)*. IEEE Computer Society, Washington, DC, USA, 365–370.

David Benavides, Pablo Trinidad, and Antonio Ruiz-Cortés. 2005. Automated reasoning on feature models. In *Proceedings of the 17th international conference on Advanced Information Systems Engineering (CAiSE'05)*. Springer-Verlag, Berlin, Heidelberg, 491–503.

Valeria Cardellini, Emiliano Casalicchio, Vincenzo Grassi, Francesco Lo Presti, and Raffaela Mirandola. 2009. Qos-driven runtime adaptation of service oriented architectures. In *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering (ESEC/FSE '09)*. ACM, New York, NY, USA, 131–140.

Jorge Cardoso, Amit P. Sheth, John A. Miller, Jonathan Arnold, and Krys Kochut. 2004. Quality of service for workflows and web service processes. *J. Web Sem.* 1, 3 (2004), 281–308.

Lawrence Chung and Julio Cesar Prado Leite. 2009. Conceptual Modeling: Foundations and Applications. Springer-Verlag, Berlin, Heidelberg, Chapter On Non-Functional Requirements in Software Engineering, 363–379.

Sholom Cohen and Krut Robert. 2010. *Managing Variation in Services in a Software Product Line Context*. Technical Report SEI-2010-TN-007. Carnegie Mellon University.

Krzysztof Czarnecki. 2005. Mapping features to models: A template approach based on superimposed variants. In *GPCE 2005 - Generative Programming and Component Enginering. 4th International Conference*. Springer, 422–437.

S.Segura D. Benavides and A. Ruiz Cortés. 2010. Automated analysis of feature models 20 years later: A literature review. *Inf. Syst.* 35, 6 (2010), 615–636.

Sybren Deelstra, Marco Sinnema, and Jan Bosch. 2009. Variability assessment in software product families. *Inf. Softw. Technol.* 51, 1 (Jan. 2009), 195–218.

Marlon Dumas, Luciano García-Bañuelos, Artem Polyvyanyy, Yong Yang, and Liang Zhang. 2010. Aggregate Quality of Service Computation for Composite Services. In *ICSOC*. 213–227.

Marlon Dumas, Jan Recker, and Mathias Weske. 2012. Management and Engineering of Process-aware Information Systems: Introduction to the Special Issue. *Information Systems* 37, 2 (2012), 77 – 79.

Leire Etxeberria and Goiuria Sagardui. 2008. Variability Driven Quality Evaluation in Software Product Lines. In *Proceedings of the 2008 12th International Software Product Line Conference (SPLC '08)*. IEEE Computer Society, Washington, DC, USA, 243–252.

F. Gottschalk, W. M.P Van Der Aalst, M. H Jansen-Vullers, and M. La Rosa. 2008. Configurable workflow models. *International Journal of Cooperative Information Systems* 17, 2 (2008), 177–221.

Caroline Herssens, Ivan J. Jureta, and Stéphane Faulkner. 2008. Capturing and Using QoS Relationships to Improve Service Selection. In *Proceedings of the 20th international conference on Advanced Information Systems Engineering (CAiSE '08)*. Springer-Verlag, Berlin, Heidelberg, 312–327.

San-Yih Hwang, Haojun Wang, Jian Tang, and Jaideep Srivastava. 2007. A probabilistic approach to modeling and estimating the QoS of web-services-based workflows. *Inf. Sci.* 177, 23 (2007), 5484–5503.

Anne Immonen. 2006. A method for predicting reliability and availability at the architecture level. In *Software Product Lines*. Springer, 373–422.

ISO/IEC. 2010. *ISO/IEC 25010 - Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models*. Technical Report.

Michael C. Jaeger, Gregor Rojec-Goldmann, and Gero Muhl. 2004. QoS Aggregation for Web Service Composition using Workflow Patterns. In *Proceedings of the Enterprise Distributed Object Computing Conference, Eighth IEEE International*. IEEE Computer Society, Washington, DC, USA, 149–159.

S. Jarzabek, B. Yang, and S. Yoeun. 2006. Addressing quality attributes in domain analysis for product lines. *Software, IEE Proceedings* 153, 2 (April 2006), 61–73.

Richard Johnson, David Pearson, and Keshav Pingali. 1994. The Program Structure Tree: Computing Control Regions in Linear Time. In *PLDI*. 171–185.

IvanJ. Jureta, Caroline Herssens, and Stphane Faulkner. 2009. A comprehensive quality model for service-oriented systems. *Software Quality Journal* 17, 1 (2009), 65–98.

Rick Kazman, Mark Klein, and Paul Clements. 2000. *ATAM: Method for Architecture Evaluation*. Technical Report CMU/SEI-2000-TR-004. Carnegie Mellon Uiversity, Software Engineering Institute.

Marc I. Kellner, Raymond J. Madachy, and David M. Raffo. 1999. Software Process Simulation Modeling: Why? What? *Journal of Systems and Software* 46 (1999), 91–105.

Bartek Kiepuszewski, Arthur H. M. ter Hofstede, and Christoph Bussler. 2000. On Structured Workflow Modelling. In *Proceedings of the 12th International Conference on Advanced Information Systems Engineering*. Springer-Verlag, London, UK, 431–445.

Kyriakos Kritikos and Dimitris Plexousakis. 2006. Semantic QoS Metric Matching. In *Proceedings of the European Conference on Web Services (ECOWS '06)*. IEEE Computer Society, Washington, DC, USA, 265–274.

Marcello La Rosa, Marlon Dumas, Arthur H. M. ter Hofstede, and Jan Mendling. 2011. Configurable multi-perspective business process models. *Inf. Syst.* 36, 2 (April 2011), 313–340.

Jaejoon Lee and Gerald Kotonya. 2010. Combining Service-Orientation with Product Line Engineering. *IEEE Software* 27 (2010), 35–41.

Kwanwoo Lee and Kyo C. Kang. 2004. Feature dependency analysis for product line component design. In *Lecture Notes in Computer Science*. Springer, 69–85.

Heiko Ludwig, Alexander Keller, Richard P. King, and Richard Franck. 2003. Web Service Level Agreement (WSLA) Language Specification. In *IBM Corporation*.

Allan MacLean, Richard M. Young, Victoria M. E. Bellotti, and Thomas P. Moran. 1991. Questions, options, and criteria: elements of design space analysis. *Hum.-Comput. Interact.* 6, 3 (Sept. 1991), 201–250.

E. Michael Maximilien and Munindar P. Singh. 2004. A Framework and Ontology for Dynamic Web Services Selection. *IEEE Internet Computing* 8, 5 (Sept. 2004), 84–93.

Ross M. McConnell and Fabien de Montgolfier. 2005. Linear-time modular decomposition of directed graphs. *Discrete Appl. Math.* 145 (January 2005), 198–209. Issue 2.

Daniel A. Menascé. 2002. QoS Issues in Web Services. *IEEE Internet Computing* 6, 6 (Nov. 2002), 72–75.

Jan Mendling. 2009. Transactions on Petri Nets and Other Models of Concurrency II. Springer-Verlag, Berlin, Heidelberg, Chapter Empirical Studies in Process Model Verification, 208–224.

Marcilio Mendonca, Moises Branco, and Donald Cowan. 2009. S.P.L.O.T.: software product lines online tools. In *Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications (OOPSLA '09)*. ACM, New York, NY, USA, 761–762.

Marcílio Mendonça, Donald D. Cowan, William Malyk, and Toacy Cavalcante de Oliveira. 2008. Collaborative Product Configuration: Formalization and Efficient Algorithms for Dependency Analysis. *JSW* 3, 2 (2008), 69–82.

Bardia Mohabbati, Mohsen Asadi, Dragan Gašević, Marek Hatala, and Hausi A. Müller. 2013b. Combining Service-Orientation and Software Product Line Engineering: A Systematic Mapping Study. *Information and Software Technology* (June 2013).

Bardia Mohabbati, Mohsen Asadi, Dragan Gašević, and Jaejoon Lee. 2013a. Software Product Line Engineering to Develop Variant-rich Web Services. In *Web Services Foundations*, Athman Bouguettaya, Quan Z. Sheng, and Florian Daniel (Eds.). Springer.

Bardia Mohabbati, Dragan Gašević, Marek Hatala, Mohsen Asadi, Ebrahim Bagheri, and Marko Bošković. 2011a. A Quality Aggregation Model for Service-Oriented Software Product Lines Based on Variability and Composition Patterns. In *ICSOC*. 436–451.

Bardia Mohabbati, Marek Hatala, Dragan Gašević, Mohsen Asadi, and Marko Bošković. 2011b. Development and configuration of service-oriented systems families. In *Proc. of the 2011 ACM Symposium on Applied Computing (SAC '11)*. ACM, New York, NY, USA, 1606–1613.

Debdoot Mukherjee, Pankaj Jalote, and Mangala Gowri Nanda. 2008. Determining QoS of WS-BPEL Compositions. In *Proceedings of the 6th International Conference on Service-Oriented Computing (ICSOC '08)*. Springer-Verlag, Berlin, Heidelberg, 378–393.

Eila Niemelä and Anne Immonen. 2007. Capturing quality requirements of product family architecture. *Inf. Softw. Technol.* 49, 11-12 (Nov. 2007), 1107–1120.

Femi G. Olumofin and Vojislav B. Mišić. 2007. A holistic architecture assessment method for software product lines. *Inf. Softw. Technol.* 49, 4 (April 2007), 309–323.

J. O'Sullivan, D. Edmond, and A. Hofstede. 2002. What's in a Service? Towards Accurate Description of Non-Functional Service Properties. *Distributed and Parallel Databases* 12, 2-3 (2002), 117–133.

Chun Ouyang, Marlon Dumas, Wil M. P. Van Der Aalst, Arthur H. M. Ter Hofstede, and Jan Mendling. 2009. From business process models to process-oriented software systems. *ACM Trans. Softw. Eng. Methodol.* 19, Article 2 (August 2009), 37 pages. Issue 1.

Ioannis V. Papaioannou, Dimitrios T. Tsesmetzis, Ioanna G. Roussaki, and Miltiades E. Anagnostou. 2006. A QoS Ontology Language for Web-Services. In *Proceedings of the 20th International Conference on Advanced Information Networking and Applications (AINA '06)*. IEEE Computer Society, Washington, DC, USA, 101–106.

Adrian Paschke. 2005. RBSLA A declarative Rule-based Service Level Agreement Language based on RuleML. In *Proc. the Int. Conf. on Computational Intelligence for Modelling, Control and Automation and Int. Conf. on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA '05)*. IEEE Computer Society, Washington, DC, USA, 308–314.

Klaus Pohl, Günter Böckle, and Frank J. van der Linden. 2005. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag New York, Inc.

Artem Polyvyanyy. 2012. *Structuring Process Models*. Dissertation. University of Potsdam.

Shuping Ran. 2003. A model for web services discovery with QoS. *SIGecom Exch.* 4, 1 (March 2003), 1–10.

Wasim Sadiq and Maria E. Orlowska. 2000. Analyzing process models using graph reduction techniques. *Information Systems* 25, 2 (2000), 117 – 134.

Norbert Siegmund, Sergiy S. Kolesnikov, Christian Kästner, Sven Apel, Don Batory, Marko Rosenmüller, and Gunter Saake. 2012. Predicting performance via automated feature-interaction detection. In *Proceedings of the 2012 International Conference on Software Engineering (ICSE 2012)*. IEEE Press, Piscataway, NJ, USA, 167–177.

Norbert Siegmund, Marko RosenmüLler, Christian KäStner, Paolo G. Giarrusso, Sven Apel, and Sergiy S. Kolesnikov. 2013. Scalable prediction of non-functional properties in software product lines: Footprint and memory consumption. *Inf. Softw. Technol.* 55, 3 (March 2013), 491–507.

Julio Sincero, Wolfgang Schroder-Preikschat, and Olaf Spinczyk. 2010. Approaching Non-functional Properties of Software Product Lines: Learning from Products. In *Proceedings of the 2010 Asia Pacific Software Engineering Conference (APSEC '10)*. IEEE Computer Society, Washington, DC, USA, 147–155.

Anil Kumar Thurimella, Bernd Bruegge, and Oliver Creighton. 2008. Identifying and Exploiting the Similarities between Rationale Management and Variability Management. In *Proceedings of the 2008 12th International Software Product Line Conference (SPLC '08)*. IEEE Computer Society, Washington, DC, USA, 99–108.

Ioan Toma, Douglas Foxvog, and Michael C. Jaeger. 2006. Modeling QoS characteristics in WSMO. In *Proceedings of the 1st workshop on Middleware for Service Oriented Computing (MW4SOC 2006) (MW4SOC '06)*. ACM, New York, NY, USA, 42–47.

Vladimir Tosic, Bernard Pagurek, Kruti Patel, Babak Esfandiari, and Wei Ma. 2005. Management applications of the web service offerings language (WSOL). *Inf. Syst.* 30, 7 (Nov. 2005), 564–586.

Vuong Xuan Tran, Hidekazu Tsuji, and Ryosuke Masuda. 2009. A new QoS ontology and its QoS-based ranking algorithm for Web services. *Simulation Modelling Practice and Theory* 17, 8 (2009), 1378–1398.

W. M. P. Van Der Aalst, A. H. M. Ter Hofstede, B. Kiepuszewski, and A. P. Barros. 2003. Workflow Patterns. *Distrib. Parallel Databases* 14 (July 2003), 5–51. Issue 1.

Jussi Vanhatalo, Hagen Völzer, and Jana Koehler. 2009. The refined process structure tree. *Data Knowl. Eng.* 68 (September 2009), 793–818. Issue 9.

Jussi Vanhatalo, Hagen Völzer, and Frank Leymann. 2007. Faster and More Focused Control-Flow Analysis for Business Process Models Through SESE Decomposition. In *Proceedings of the 5th international conference on Service-Oriented Computing (ICSOC '07)*. Springer-Verlag, Berlin, Heidelberg, 43–55.

Asir S Vedamuthu, David Orchard, Frederick Hirsch, Maryann Hondo, Prasad Yendluri, Toufic Boubez, and mit Yalinalp. 2007. Web Services Policy Framework (WSPolicy). (September 2007). http://www.w3.org/TR/ws-policy

Xia Wang, Tomas Vitvar, Mick Kerrigan, and Ioan Toma. 2006. A qos-aware selection model for semantic web services. In *Proceedings of the 4th international conference on Service-Oriented Computing (ICSOC'06)*. Springer-Verlag, Berlin, Heidelberg, 390–401.

Yong Yang, Marlon Dumas, Luciano García-Bañuelos, Artem Polyvyanyy, and Liang Zhang. 2012. Generalized aggregate Quality of Service computation for composite services. *J. Syst. Softw.* 85, 8 (Aug. 2012), 1818–1830.

Tao Yu and Kwei jay Lin. 2005. Service selection algorithms for composing complex services with multiple qos constraints. In *ICSOC*. 130–143.

Yijun Yu, Julio Cesar Sampaio do Prado Leite, Alexei Lapouchnian, and John Mylopoulos. 2008. Configuring features with stakeholder goals. In *Proceedings of the 2008 ACM symposium on Applied computing (SAC '08)*. ACM, New York, NY, USA, 645–649.

L. Zeng, B. Benatallah, A.H.H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. 2004. QoS-aware middleware for web services composition. *Software Engineering, IEEE Transactions on* 30, 5 (2004), 311–327.

Hongyu Zhang, Stan Jarzabek, and Bo Yang. 2003. Quality prediction and assessment for product lines. In *Proceedings of the 15th international conference on Advanced information systems engineering (CAiSE'03)*. Springer-Verlag, Berlin, Heidelberg, 681–695.

Zibin Zheng, Yilei Zhang, and Michael R. Lyu. 2010. Distributed QoS Evaluation for Real-World Web Services. In *Proceedings of the 2010 IEEE International Conference on Web Services (ICWS '10)*. IEEE Computer Society, Washington, DC, USA, 83–90.

Received x; revised x; accepted x