A Quality Aggregation Model for Service-Oriented Software Product Lines Based on Variability and Composition Patterns

Bardia Mohabbati[1], Dragan Gašević[1,2], Marek Hatala[1], Mohsen Asadi[1], Ebrahim Bagheri[2,3], and Marko Bošković[1,2]

[1] Simon Fraser University, Canada
{mohabbati,mhatala,masadi}@sfu.ca
[2] Athabasca University, Canada
{dragang,ebagheri,marko.boskovic}@athabascau.ca
[3] University of British Columbia, Canada

Abstract. Quality evaluation is a challenging task in monolithic software systems. It is even more complex when it comes to Service-Oriented Software Product Lines (SOSPL), as it needs to analyze the attributes of a *family* of SOA systems. 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 non-functional quality attribute satisfaction. Therefore, architectural quality evaluation becomes crucial due to the fact that it allows for the examination of whether or not 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 QoS attribute 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 an aggregation model for QoS computation which takes both variability and composition patterns into account.

Keywords: Software Product Line (SPL), Service-Oriented Architecture (SOA), non-functional properties, QoS aggregation, process family, service variability, variability management, feature modeling.

1 Introduction

The Service-Oriented Architecture (SOA) paradigm enables the realization of Software-as-a-Service (SaaS). Some service providers have already moved towards the adoption of customizable product-development 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 produce new applications systematically [1]. *Software Product Line Engineering (SPLE)* provides a platform to capture both functional and non-functional aspects and allows for the rapid customization of new products. Several researchers have proposed to integrate SOA and SPLE paradigms into *Service-Oriented Software Product Lines (SOSPLs)* as a way

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 436–451, 2011.

to formalize *customizable product-development* and take the benefits and synergies of both paradigms [1,2].

Researchers have explored various strategies for the realization of software applications, e.g., how the most appropriate services can be selected for a given product line and how they can be efficiently composed. However, previous works often fails to consider Quality-of-Service (QoS) in the context of product lines. Quality evaluation is a challenging task in monolithic software systems and it is even more complex when it comes to SOSPL, as it needs to analyze the attributes of a family of SOA systems.

This paper contributes a solution to the following open research problem: *How can the quality attributes of a software product line 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 QoS properties at the architectural level. Building on our previous work [3,4], we assume that QoS dimensions are captured in terms of quantitative properties. In particular, this paper makes the following contributions:

1. The introduction and classification of 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 modeling;
2. The development of a *quality model framework* for the aggregation of QoS based on different architectural patterns;
3. The formalization of a *computational model* for architectural quality evaluation, which takes into account both variability and composition patterns and allows for tradeoff analysis and architectural decision making between options that provide similar functional properties but different levels of quality.

The reminder of this paper is organized as follow: Section 2 describes SOSPL and its related conceptual modeling and formalism. QoS aggregation and computation model for SOSPL architecture is described in Section 3. The discussion and complexity evaluation of methods is presented in Section 4. The related work is discussed in Section 5. Finally, Section 6 presents the conclusion and future work.

2 Service-Oriented Software Product Lines

SPLE has been recognized as a successful approach to variability management and reuse engineering, which enables mass customization, enhances software quality and reduces the time-to-market of new software products [5]. 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 [5]. 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* [5]. Domain Engineering is concerned with the analysis and identification of

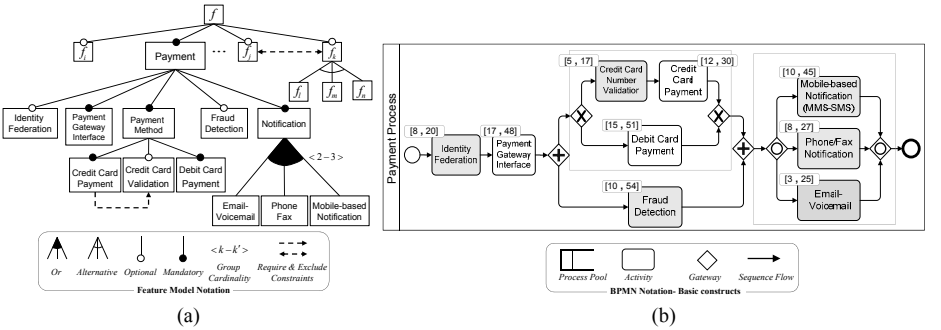
the scope of the product line and the capturing of the entire domain of interest through modeling of common and variation points. An Application Engineering cycle builds the understanding of specific requirements of different stakeholders, for whom the customization and configuration of the product line is carried out. We have presented the details of these two distinctive lifecycles in [3].

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) [1,2]. Such approaches benefit from SOA principles to provide an actual implementation of SPL products. Let us review an illustrative example in this regard.

2.1 Illustrative Example

In Fig.1, we show a simplified business process model for e-Payment in a global online retailer scenario to illustrate the concepts and further discussions. As shown, different features from the feature model (on the left) that can be used within the business process model (on the right) to implement the functionality of the payment process. These features can be realized using appropriate services, which can have different QoS characteristics. For example, 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 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 feature model such as the one in Fig.1(a) is a model of a family of products (SPL), while each variant (customized service) is a member of that family. Variation points are



(a) (b)

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

those places in the design of the SOSPL architecture where a specific decision has been narrowed down to several options. However, the options to be selected for a particular application w.r.t. stakeholder's requirements are left open for configuration. Hence, the 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 to 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 properties of a feature model based on the QoS characteristics 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 products that will be derived from that product family. Furthermore, 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. Therefore, the main contribution of this work is an introduction of the process how these QoS ranges are computed in the presence of variability.

2.2 Feature Modeling

Feature modeling is one of the important techniques for capturing, modeling and describing the commonalities and differences between the products of a family based on their features. A feature model is a means for describing a permissible configuration space of all the products of a family in terms of its features and their relationships. A feature model consists of both formal semantics and graphical representation, which is a rooted directed acyclic graph (DAG). Fig.1(a) depicts a part of the feature model in our example, which describes common and variable features and represents architectural variability in the reference business 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 *architectural variability patterns* as follows:

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

We formally define a feature model as follows:

Definition 1 (Feature Model). *A feature model* $FM{=}G_{FM}(V, E)$ *is a directed acyclic graph consisting of the set of vertices* V *representing features and edges* $E \subseteq V{\times}V$ *representing the parent-child relations between the features, such that* $E{=}\{\overset{\bullet}{f}, \overset{\circ}{f}, f_{or}, f_{xor}\}$,

where $\overset{\bullet}{f}$ *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.*

In general, *cardinality* is also defined among a group of $n > 1$ sub-features, which is denoted by $<k-k'>$, where $1 \leq k \leq k' \leq n$. Hence, $<k-k'>$ cardinality defined over *Or feature groups* indicates at least k and at most k' features can be included out of the n features in a group if the parent is selected. In addition, *integrity constraints*, i.e., the *includes and excludes* relations, can be defined over features of a feature model. They are the means to express that the presence of a certain feature in the product imposes the presence or exclusion of another feature.

As it can be observed, except for the mandatory feature type, all the other types of features imply architectural variability.

2.3 Reference Business Process Model

A template-based approach, where a reference model is designed as a template and is further customized for various purposes, has been widely adopted by practitioners [6]. The reference model contains a union of the business processes for the entire product line in a superimposed way. The reference model provides the common business logic for orchestration and choreography of services. The design of reference models is accomplished in the course of the domain engineering lifecycle [5].

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 in variation points, which are managed and configured by means of feature models. It should be noted that we distinguish between design and runtime variability. 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).

We consider a business process model as a workflow which is formally defined as follows:

Definition 2 (Business Process Model). *Business process model BP is defined as a directed acyclic graph* $G_{BP} = (V, E)$*, where* $V = \{n_\epsilon, V_\sigma, V_A, V_G\}$ *denotes a set of disjoint nodes,* n_ϵ *is a unique initial state,*V_σ *is a final state,* V_A *is a set of activities, and* E *represents the edges (transitions) between the nodes.* V_G *is a set of nodes as gateways.*

A business process is viewed as a series of activities where an activity represents a functional abstraction of services. An activity can be 1) atomic (a.k.a., task) or 2) non-atomic (a.k.a., sub-process). Each activity is delegated and bound to one or more services that provide the required functionality with different quality properties. Gateways, as routing constructs, represent a control flow of branchings, i.e., routing points. In this paper, we impose the following well-formedness conditions on a business process structure [7]: i) a business process model has a single source node, i.e., a node with no incoming

edge; ii) every activity node has a single incoming and a single outgoing edge; iii) for every node with multiple outgoing arcs (i.e., a split), there is a corresponding node with multiple incoming arcs (a join), such that the set of nodes between the split and the join forms a single-entry-single-exit region [8].

In our work, architectural and behavioral variability are described by means of two models, i.e., feature models and business process models, respectively. Hence, we will assume that there is a mapping model available which interconnects these two models [3,6]. This injective mapping (i.e., one-to-one) reciprocally links each feature in the feature model to the corresponding activity in the reference business process model.

3 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 in order to provide a model for aggregating and computing QoS range values in the presence of variability: 1) quality criteria and quality range values for SOSPL; 2) the combination of variability and composition patterns; 3) based on the combination of variability and composition patterns; and 4) computational algorithms for computing aggregate quality range values.

3.1 Quality Criteria for Service-Oriented Product Line

Different fields of research and standards have proposed diverse definitions and ontologies for describing QoS properties based on their target application domains [9].

We consider some quantitative QoS characteristics of Web services, which have been taken into consideration as selection criteria in the research literature [10,9,11,12]. Specifically, *cost* and *response time* will be the two indexed QoS properties included in our work, which will be denoted by q_{pr} and q_{rt}, respectively, throughout the paper. Of course, our approach is not limited to these two QoS types, but these are the only ones discussed here due the limited space of the paper.

Let us now proceed with some formal definitions as a basis for our work.

Definition 3 (Quality Range). *The quality range values of the i^{th} quality property (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 property, 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 property. 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* [9,13] 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 $a_n \in V_A$ 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 QoS properties. 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 i^{th} quality property.

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 property q_i, while each column corresponds to a service candidate. Thereby, the range of the i^{th} quality property 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 as follows: $q_{pr}^{\mathrm{R}}(f_k) = [65, 250]$, because we are interested in the lower and upper bound values for the quality range.

3.2 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[14], 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. They represent the abstract control flow and execution sequence of features within the reference business process model for the whole family. Similar to [12], we consider composition patterns that address the behavioral structure of a composition. These patterns can be grouped into two main groups: a) *sequential patterns* and b) *parallel patterns*. These patterns are defined in terms of how the process flow proceeds in sequences and splits into branches for executing the activities and how they merge or converge. For our work, we consider the combination of parallel split, convergence and synchronization patterns.

Fig.2 illustrates three variability patterns (left side), as described in Sec.2.2, in combination with nine composition patterns (CP1-CP9) represented using BPMN notation (right side). We perform quality aggregation based on a set of proposed aggregation rules described below.

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

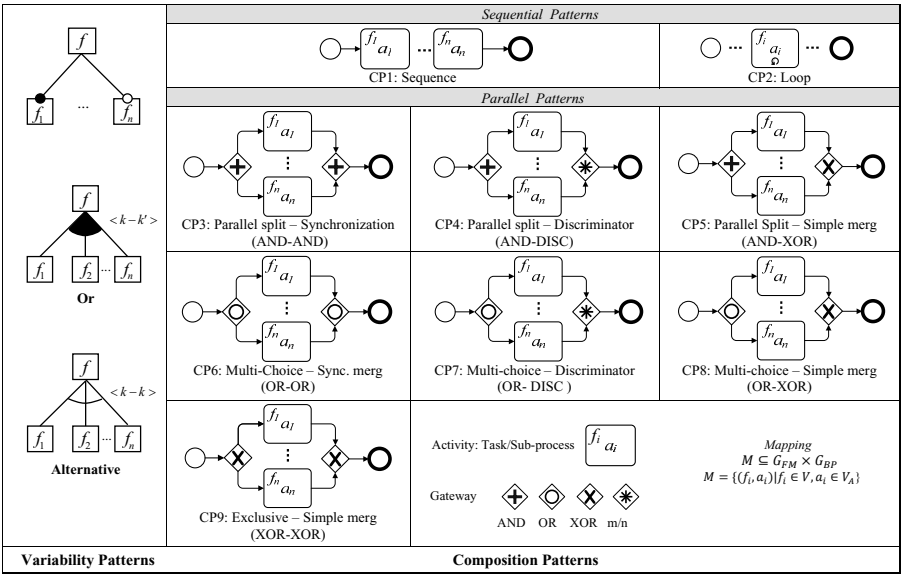


Fig. 2. Variability and composition patterns

3.3 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 two numerical QoS properties: cost and response time (i.e., execution time). The cost of a feature is the cost which can be associated with the deployment, execution, management, maintenance and monitoring of a service. The aggregation rules for availability and throughput are presented in a longer version of the current paper which is accessible online[2]. The summary of the aggregation rules that we have defined are given in Tables 1 and 2.

According to Definition 3, the definition of a lower bound, q_i^{LB}, for different quality properties must indispensably consider the mandatory features for sequential and parallel split patterns (CP1-CP4). 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 feature f should be determined by the sum of the cost values of each mandatory feature for the lower bound; while the upper bound is determined by the accumulated cost for mandatory as well as optional features.

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

[2] `http://qos-sospl.sourceforge.net/`

Table 1. Aggregation rules based on *Mandatory-Optional* variability patterns

Mandatory-Optional Variability Pattern		QoS Properties		Cost (q_c)	Response Time (q_{rt})
Seq. Patterns	1	Sequence	$\left[\sum_{i=1}^{n} q_{pr}^{LB}(f_i):\forall f_i \in \dot{f} \;,\; \sum_{i=1}^{n} q_{pr}^{UB}(f_i):\forall f_i \in \dot{f}\vee\hat{f}\right]$	$\left[\sum_{i=1}^{n} q_{pr}^{LB}(f_i):\forall f_i \in \dot{f} \;,\; \sum_{i=1}^{n} q_{pr}^{UB}(f_i):\forall f_i \in \dot{f}\vee\hat{f}\right]$
	2	Arbitrary Cycle	$\left[cq_{pr}^{LB}(f_i): f_i \in \dot{f}\vee\hat{f} \;,\; cq_{pr}^{UB}(f_i): f_i \in \dot{f}\vee\hat{f}\right]$	$\left[cq_{rt}^{LB}(f_i): f_i \in \dot{f}\vee\hat{f} \;,\; cq_{rt}^{UB}(f_i): f_i \in \dot{f}\vee\hat{f}\right]$
Parallel Patterns	3	AND-AND	$\left[\sum_{i=1}^{n} q_{pr}^{LB}(f_i):\forall f_i \in \dot{f} \;,\; \sum_{i=1}^{n} q_{pr}^{UB}(f_i):\forall f_i \in \dot{f}\vee\hat{f}\right]$	$\left[\max\left(q_{rt}^{LB}(f_i):\forall f_i \in \dot{f}\right) \;,\; \max\left(q_{rt}^{UB}(f_i):\forall f_i \in \dot{f}\vee\hat{f}\right)\right]$
	4	AND-DISC		$\left[\min\left(q_{rt}^{LB}(f_i):\forall f_i \in \dot{f}\right) \;,\; \max\left(q_{rt}^{UB}(f_i):\forall f_i \in \dot{f}\vee\hat{f}\right)\right]$
	5	AND-XOR		
	6	XOR-XOR	$\left[\min\left(q_{pr}^{LB}(f_i): f_i \in \dot{f}\vee\hat{f}\right) \;,\; \max\left(q_{pr}^{UB}(f_i): f_i \in \dot{f}\vee\hat{f}\right)\right]$	$\left[\min\left(q_{rt}^{LB}(f_i):\forall f_i \in \dot{f}\vee\hat{f}\right) \;,\; \max\left(q_{rt}^{UB}(f_i):\forall f_i \in \dot{f}\vee\hat{f}\right)\right]$
	7	OR-XOR		
	8	OR-OR	$\left[\min\left(\sum_{f_i \in F_{Sub}} q_{pr}^{LB}(f_i):\forall F_{Sub} \in F_{C_m^n}\right) \;,\; \max\left(\sum_{f_i \in F_{Sub}} q_{pr}^{UB}(f_i):\forall F_{Sub} \in F_{C_m^n}\right)\right]$	$\left[\min\left(\max\left(q_{pr}^{LB}(f_i): f_i \in F_{Sub}, \forall F_{Sub} \in F_{C_m^n}\right)\right)^{1} \;,\; \max\left(q_{rt}^{UB}(f_i):\forall f_i \in \dot{f}\vee\hat{f}\right)\right]$
	9	OR-DISC		

[1]For example, assuming three features (m =3) out of seven (n=7) in CP6-CP7, the minimum of 3rd quickest should be considered

f (see Fig.2), the aggregation rules are applied on the basis of each composition pattern according to the variability patterns.

To further introduce the principles of our aggregation model, the following explanation is provided. Feature set $F_{C_l^n}$ contains all of the permissible combinations of the feature sets, where the number of distinct l-element subsets is a binomial coefficient denoted as C_l^n. 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. CP6, CP7 and CP8) follow different aggregation rules. To determine the lower and upper bounds, the aggregation model also requires knowing which paths in the business process flow will be chosen at runtime particularly for OR-Splits. We assume that an 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 runtime. 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 runtime.

To address *Or group* and *Alternative group* feature variability in combination with Multi and Exclusive choice composition patterns (CP7, CP8, and CP9), 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.

3.4 Quality of Service Range Aggregation

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

Table 2. Aggregation rules based on *Or* and *Alternative* variability patterns

Or[*] / *Alternative*[**] *Variability Patterns*			QoS Properties	Cost (q_c)	Response Time (q_{rt})
	Seq. Pattern	1	Sequence	$\left[\min\left(\sum_{f_i \in F_{Stub}} q_{pr}^{LB}(f_i) : \forall F_{Stub} \in F_{C_k^n}\right), \max\left(\sum_{f_i \in F_{Stub}} q_{pr}^{UB}(f_i) : \forall F_{Stub} \in F_{C_{k'}^n}^{(*)} \middle| F_{C_k^n}^{(**)}\right)\right]$	$\left[\min\left(\sum_{f_i \in F_{Stub}} q_{rt}^{LB}(f_i) : \forall F_{Stub} \in F_{C_k^n}\right), \max\left(\sum_{f_i \in F_{Stub}} q_{rt}^{UB}(f_i) : \forall F_{Stub} \in F_{C_{k'}^n}^{(*)} \middle| F_{C_k^n}^{(**)}\right)\right]$
	Parallel Patterns	3	AND-AND	$\left[\min\left(\sum_{f_i \in F_{Stub}} q_{pr}^{LB}(f_i) : \forall F_{Stub} \in F_{C_k^n}\right), \max\left(\sum_{f_i \in F_{Stub}} q_{pr}^{UB}(f_i) : \forall F_{Stub} \in F_{C_{k'}^n}^{(*)} \middle| F_{C_k^n}^{(**)}\right)\right]$	$\left[\min\left(\max\left(q_{rt}^{LB}(f_i) : \forall F_{Stub} \in F_{C_k^n}\right)\right), \max\left(\max\left(q_{rt}^{UB}(f_i) : \forall F_{Stub} \in F_{C_{k'}^n}^{(*)} \middle| F_{C_k^n}^{(**)}\right)\right)\right]$
		4	AND-DISC		$\left[\min\left(\min\left(q_{rt}^{LB}(f_i) : \forall F_{Stub} \in F_{C_k^n}\right)\right), \max\left(\max\left(q_{rt}^{UB}(f_i) : \forall F_{Stub} \in F_{C_{k'}^n}^{(*)} \middle| F_{C_k^n}^{(**)}\right)\right)\right]$
		5	AND-XOR		
		6	OR-XOR		
		7	OR-OR		
		8	OR-DISC		$\left[\min\left(\max\left(q_{rt}^{LB}(f_i) : \forall F_{Stub} \in F_{C_k^n}\right)\right), \max\left(\max\left(q_{rt}^{UB}(f_i) : \forall F_{Stub} \in F_{C_{k'}^n}^{(*)} \middle| F_{C_k^n}^{(**)}\right)\right)\right]$
		9	XOR-XOR		

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

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.

Algorithm 1. Aggregate QoS range for feature: AggregateQoSRange(f)

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

```
1  begin
       // All direct child features of f;
2      S_f[ ] ⟵ ∀ChildFeatureOf(f) ;
3      if S_f = ∅ then return q^R(f);
4      else
5          for i = 1 to |S_f| do
6              AggregateQoSRange(S_f[i]);
7              PST ⟵ ProcessStructure(S_f[i]);
8              if PST = ∅ then return q^R(f) else q^R(f)= ComputeQoSR(PST)
9
10 end
```

Algorithms 1 and 2 detail the procedure for computing QoS ranges for a feature model. In these algorithms, 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. For each feature, the business Process Structure Tree (PST) corresponding to a given feature is subsequently created and parsed (Fig.3(b)). For every trivial Single-Entry Single-Exit (SESE) component in PST, the control flow analysis is performed, whose details are omitted from Alg. 2 for the sake of brevity; interested readers are referred to [15].

In order to analyze and identify how features at the same level in the feature model are composed, we decompose the process graph into process components (Fig. 3). A process component is a subgraph of the composition model with a SESE region, which may include individual tasks but also larger subgraphs. We employ the Refined Process Structure Tree (RPST)-based approach proposed in [13,8] to create and parse the process model into a tree of SESE components (line 7 in Alg. 1).

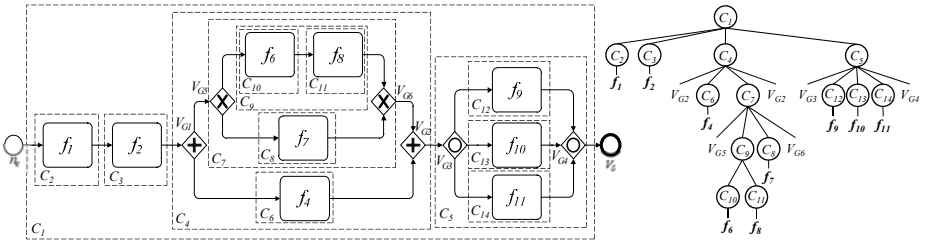
Alg. 2 operates over a PST, and the aggregation is performed at the level of each processes component. Lines 3 to 18 iteratively aggregate the quality for each child component. For individual process components, features are grouped into virtual features corresponding to the variability patterns.

Algorithm 2. Compute QoS range values of business process associated to feature f: ComputeQoSR(C)

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

1 **begin**
2 **foreach** $C_i \in ChildOf(C)$ **do** ComputeQoSR(C_i)
3 **forall** $f_k \in C_i$ **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** $f_k \in \overset{\circ}{f} \vee \overset{\bullet}{f}$
6 $f_{V_{mo}}[\,] \longleftarrow f_k$;
7 $q^{\mathrm{R}}(f_{V_{mo}}) = \mathrm{AggQoS}(f_{V_{mo}})$ w.r.t. Formulas in Table 1;
8 **if** $\forall f_k \in f_{V_{mo}} : f_k \in \overset{\circ}{f}$ **then** $f_{V_{mo}}$.*Type*=$\overset{\circ}{f}$ **else** $f_{V_{mo}}$.*Type*=$\overset{\bullet}{f}$;
9 **case** $f_k \in$ *an Or-group*
10 $\overset{\bullet}{f}_{V_{or}}[\,] \longleftarrow f_k$;
11 $q^{\mathrm{R}}(\overset{\bullet}{f}_{V_{or}}) = \mathrm{AggQoS}(\overset{\bullet}{f}_{V_{or}})$ w.r.t. Formulas in Table 2;
12 **case** $f_k \in$ *an Alternative-group*
13 $\overset{\bullet}{f}_{V_{xor}}[\,] \longleftarrow f_k$;
14 $q^{\mathrm{R}}(\overset{\bullet}{f}_{V_{xor}}) = \mathrm{AggQoS}(\overset{\bullet}{f}_{V_{xor}})$ w.r.t. Formulas in Table 2;
15 **end**
16 $f_V[\,] \longleftarrow f_{V_{mo}}, \overset{\bullet}{f}_{V_{or}}, \overset{\bullet}{f}_{V_{xor}}$;
17 $q^{\mathrm{R}}(f_V) = \mathrm{AggQoS}(f_V)$ w.r.t. Formulas given in Tables 1,2;
18 **if** $\exists f_k \in f_V : f_i \in \overset{\bullet}{f}$ **then** f_V.*Type*=$\overset{\bullet}{f}$ **else** f_V.*Type*=$\overset{\circ}{f}$;
19 **end**
20 **return** $q^{\mathrm{R}}(f_V)$
21 **end**

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*, *Or* and *Alternative* grouped virtual features, respectively. Quality range values of virtual features are computed by an aggregation function (AggQoS) according to the aggregation rules introduced earlier in the paper and shown in Tables 1 and 2. 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 8). However, it is noted that according to the given semantic descriptions of feature variability of *Or* and *Alternative groups*, corresponding virtual



(a) (b)

Fig. 3. a) Decomposition of business process model to SESE components b) Business process structure tree

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 18).

To exemplify the aggregation algorithm described above, we compute the QoS range values of cost (q_{pr}) for the payment feature in the feature model shown in Fig.1. Fig.4 depicts the step-wise transformation of the feature model through gradual hierarchical aggregations. The following is also 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 algorithm that is used besides each step.

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

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

$$q^{\mathrm{R}}_{pr}(\overset{\bullet}{f}_{V_1}) = \underbrace{\{q^{\mathrm{R}}_{pr}(\overset{\circ}{f}_6), q^{\mathrm{R}}_{pr}(\overset{\bullet}{f}_8)\}}_{\overset{1}{\bowtie}} = \underbrace{\left[q^{\mathrm{LB}}_{pr}(\overset{\bullet}{f}_8), \sum\{q^{\mathrm{UB}}_{pr}(\overset{\bullet}{f}_8), q^{\mathrm{UB}}_{pr}(\overset{\circ}{f}_6)\}\right]}_{\text{Tablel-Agg.rule No.1}} = [12,47]; \quad \text{(Alg.2 lines 5-8)}$$

$$q^{\mathrm{R}}_{pr}(\overset{\bullet}{f}_3) = \underbrace{\{q^{\mathrm{R}}_{pr}(\overset{\bullet}{f}_{V_1}), q^{\mathrm{R}}_{pr}(\overset{\bullet}{f}_7)\}}_{\overset{2}{\bowtie}} = \underbrace{\left[\min\left(\{q^{\mathrm{LB}}_{pr}(\overset{\bullet}{f}_{V_1}), q^{\mathrm{LB}}_{pr}(\overset{\bullet}{f}_7)\}\right), \max\left(\{q^{\mathrm{UB}}_{pr}(\overset{\bullet}{f}_{V_1}), q^{\mathrm{UB}}_{pr}(\overset{\circ}{f}_7)\}\right)\right]}_{\text{Tablel-Agg.rule No.6}} \quad \text{(Alg.2 lines 5-8)}$$

$$= [12,51];$$

$$q^{\mathrm{R}}_{pr}(\overset{\bullet}{f}_{V_2}) = \underbrace{\{q^{\mathrm{R}}_{pr}(\overset{\bullet}{f}_3), q^{\mathrm{R}}_{pr}(\overset{\circ}{f}_4)\}}_{\overset{3}{\bowtie}} = \underbrace{\left[q^{\mathrm{LB}}_{pr}(\overset{\bullet}{f}_3), \sum\{q^{\mathrm{UB}}_{pr}(\overset{\bullet}{f}_3), q^{\mathrm{UB}}_{pr}(\overset{\circ}{f}_4)\}\right]}_{\text{Table 1.Agg. rule No. 3}} = [12,105]; \quad \text{(Alg.2 lines 5-8)}$$

$$q^{\mathrm{R}}_{pr}(\overset{\bullet}{f}_5) = \underbrace{\left[\min\left(\sum_{f_i \in F_{Sub}} q^{\mathrm{LB}}_{pr}(f_i) : \forall F_{Sub} \in F_{C^3_2}\right), \max\left(\sum_{f_i \in F_{Sub}} q^{\mathrm{UB}}_{pr}(f_i) : \forall F_{Sub} \in F_{C^3_3}\right)\right]}_{\overset{4}{\bowtie}\ \text{Table 2.Agg. rule No.7}} \quad \text{(Alg.2 lines 9-11)}$$

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

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

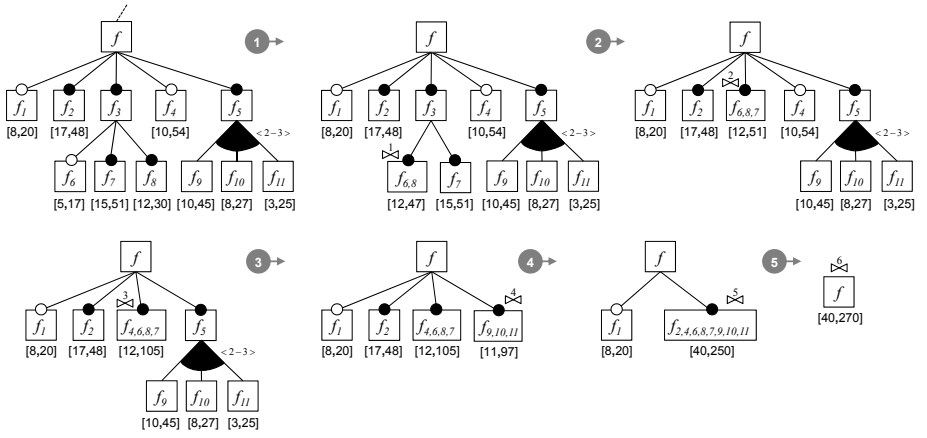


Fig. 4. Step-wise feature model transformation

$$= \underbrace{\left[\sum\{q_{pr}^{\mathrm{LB}}(\overset{\bullet}{f}_2), q_{pr}^{\mathrm{LB}}(\overset{\bullet}{f}_{V_2})\}\ , \sum\{q_{pr}^{\mathrm{UB}}(\overset{\bullet}{f}_2), q_{pr}^{\mathrm{UB}}(\overset{\circ}{f}_5), q_{pr}^{\mathrm{UB}}(\overset{\bullet}{f}_{V_2})\}\right]}_{\text{Table 1.Agg. rule No. 1}} = [40, 250];$$

$$q_{pr}^{\mathrm{R}}(f) = \underbrace{\{q_{pr}^{\mathrm{R}}(\overset{\circ}{f}_1), q_{pr}^{\mathrm{R}}(\overset{\bullet}{f}_{V_3})\}}_{\bowtie^6} = \underbrace{\left[q_{pr}^{\mathrm{LB}}(\overset{\bullet}{f}_{V_3}), \sum\{q_{pr}^{\mathrm{UB}}(\overset{\bullet}{f}_2), q_{pr}^{\mathrm{UB}}(\overset{\circ}{f}_5), q_{pr}^{\mathrm{UB}}(\overset{\bullet}{f}_{V_2})\}\right]}_{\text{Table 1.Agg. rule No.1}} = [40, 270]$$

It should be noted that the cost represents a QoS property in this example, which follows certain aggregation rules as described formerly; however, different rules are applied for aggregation of range values for other QoS properties, e.g., response time.

4 Discussion

In this section, we analyze the computational complexity of the proposed aggregation method, and then critically discuss its advantages and disadvantages.

4.1 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 if the number of integrity constraints on a feature model is too high. 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 [16]). 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 algorithm has a linear time complexity when the number of integrity constraints is smaller than the number of features in a feature model, which is usually the case based on our analysis of standard feature models available at `http://www.splot-research.org/`, where the number of integrity constraints to the number of features ratio is approximately 0.18.

4.2 Critical Analysis

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 [7]. However, such an assumption requires the transformation 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 [13,17,8,7]. 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 aggregation of QoS 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. Finally, this work can be further considered for facilitating the management and customization of multi-tenant cloud applications [18].

5 Related Work

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. [11] and Jaeger et al. [12] are the seminal works in the literature, 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. [14]. In [11], the authors propose 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. In [12], where the aggregation method is the most similar to ours, a QoS aggregation method is proposed for composite Web services 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. [19] have 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. In a more recent work [13], an aggregation approach employs RPST and supports process models which include unstructured components.

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 for evaluating QoS dimensions in the context of SOSPL.

6 Conclusion

In this paper, we have provided a systematic approach for QoS range aggregation to support evaluation of quality ranges captured by SOSPL(s), which further helps us for quality-aware product derivation. As the main contribution, we have identified a set of variability patterns which may occur within composition patterns and proposed new aggregation rules for QoS range computation. We also presented an algorithm that analyzes variability and process models to aggregates QoS ranges for an SOSPL. The present work is a continuation of our previous works [3,4], where we have presented approaches for configuration of SOSPLs in the application engineering lifecyle. In those works, we also proposed a method for features prioritization based on stakeholders' objectives and preferences concerning functional and QoS requirements. We also presented how that method can be leveraged by using linear optimization methods for optimal service selection within boundaries of constraints specified by the stakeholders. The approach presented in this paper takes our previous work to next stage where automatic computation of quality ranges in the presence of variability is made possible. As the future work, we also intend to evaluate how our proposed approaches for configuration enabled by the contribution of this work can be applied to real-world scenarios.

References

1. Cohen, S.G., Krut, R.: Managing variation in services in a software product line context. Technical Report SEI-2010-TN-007, Carnegie Mellon University (2010)
2. Lee, J., Kotonya, G.: Combining service-orientation with product line engineering. IEEE Software 27, 35–41 (2010)
3. Mohabbati, B., Hatala, M., Gašević, D., Asadi, M., Bošković, M.: Development and configuration of service-oriented systems families. In: Proceedings of the 2011 ACM Symposium on Applied Computing, SAC 2011, pp. 1606–1613. ACM, New York (2011)
4. Bagheri, E., Asadi, M., Gasevic, D., Soltani, S.: Stratified Analytic Hierarchy Process: Prioritization and Selection of Software Features. In: Bosch, J., Lee, J. (eds.) SPLC 2010. LNCS, vol. 6287, pp. 300–315. Springer, Heidelberg (2010)
5. Pohl, K., Böckle, G., van der Linden, F.J.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer-Verlag New York, Inc. (2005)
6. Czarnecki, K., Antkiewicz, M.: Mapping Features to Models: A Template Approach Based on Superimposed Variants. In: Glück, R., Lowry, M. (eds.) GPCE 2005. LNCS, vol. 3676, pp. 422–437. Springer, Heidelberg (2005)
7. Kiepuszewski, B., ter Hofstede, A.H.M., Bussler, C.J.: On Structured Workflow Modelling. In: Wangler, B., Bergman, L.D. (eds.) CAiSE 2000. LNCS, vol. 1789, pp. 431–445. Springer, Heidelberg (2000)
8. Vanhatalo, J., Völzer, H., Koehler, J.: The refined process structure tree. Data Knowl. Eng. 68, 793–818 (2009)
9. Zeng, L., Benatallah, B., Ngu, A., Dumas, M., Kalagnanam, J., Chang, H.: QoS-aware middleware for web services composition. IEEE Transactions on Software Engineering 30, 311–327 (2004)
10. Yu, T., Lin, K.-J.: Service Selection Algorithms for Composing Complex Services with Multiple qoS Constraints. In: Benatallah, B., Casati, F., Traverso, P. (eds.) ICSOC 2005. LNCS, vol. 3826, pp. 130–143. Springer, Heidelberg (2005)
11. Cardoso, J., Sheth, A.P., Miller, J.A., Arnold, J., Kochut, K.: Quality of service for workflows and web service processes. J. Web Sem. 1, 281–308 (2004)
12. Jaeger, M.C., Rojec-Goldmann, G., Muhl, G.: Qos aggregation for web service composition using workflow patterns. In: Proceedings of the Eighth IEEE International Conference on Enterprise Distributed Object Computing, pp. 149–159. IEEE Computer Society, Washington, DC, USA (2004)
13. Dumas, M., García-Bañuelos, L., Polyvyanyy, A., Yang, Y., Zhang, L.: Aggregate Quality of Service Computation for Composite Services. In: Maglio, P.P., Weske, M., Yang, J., Fantinato, M. (eds.) ICSOC 2010. LNCS, vol. 6470, pp. 213–227. Springer, Heidelberg (2010)
14. Van Der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow patterns. Distrib. Parallel Databases 14, 5–51 (2003)
15. Vanhatalo, J., Völzer, H., Leymann, F.: Faster and More Focused Control-Flow Analysis for Business Process Models Through SESE Decomposition. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 43–55. Springer, Heidelberg (2007)
16. McConnell, R.M., de Montgolfier, F.: Linear-time modular decomposition of directed graphs. Discrete Appl. Math. 145, 198–209 (2005)
17. Ouyang, C., Dumas, M., van der Aalst, W.M.P., ter Hofstede, A.H.M., Mendling, J.: From business process models to process-oriented software systems. ACM Trans. Softw. Eng. Methodol. 19, 2:1–2:37 (2009)
18. van der Aalst, W.M.P.: Configurable Services in the Cloud: Supporting Variability While Enabling Cross-Organizational Process Mining. In: Meersman, R., Dillon, T.S., Herrero, P. (eds.) OTM 2010. LNCS, vol. 6426, pp. 8–25. Springer, Heidelberg (2010)
19. Hwang, S.Y., Wang, H., Tang, J., Srivastava, J.: A probabilistic approach to modeling and estimating the qos of web-services-based workflows. Inf. Sci. 177, 5484–5503 (2007)