An Architecture-Centric Development Approach for Service-Oriented Product Lines

Xingjian Lu[1,3], Jianwei Yin[4], Gaoqi He[1], Huiqun Yu[2], Neal N. Xiong[5]
[1] College of Computer Science and Technology, East China Normal University, China
[2] School of Information Science and Engineering, East China University of Science and Technology, China
[3] Smart City Collaborative Innovation Center, Shanghai Jiao Tong University, China
[4] College of Computer Science and Technology, Zhejiang University, China
[5] Dept. of Mathematics and Computer Science, Northeastern State University, USA
luxj@ecust.edu.cn, zjuyjw@zju.edu.cn, hegaoqi@sei.ecnu.edu.cn, yhq@ecust.edu.cn, xiongnaixue@gmail.com

Abstract

Service-Oriented Product Line (SOPL), which combines Service-Oriented Architecture (SOA) and Software Product Line (SPL) concepts and technologies, has attained an increasing interest in software engineering community in recent years. However, there are still several challenges we have to overcome when developing a SOPL. In this paper, an architecture-centric approach for SOPL development is proposed to alleviate these challenges. First, the business process execution language (BPEL) based architecture style and architecture description language bpel4Arch are developed. Then, based on them, a model-driven reference architecture generating method and a common rule engine based architecture customization approach are proposed. For facilitating the translation between application architecture and BPEL, we also provide an optimal partition for application architecture by applying the mixed integer programming (MIP) technique, which can achieve the entire function of target product with minimum molecular services. Finally, an initial case study on E-Shopping domain and some evaluations show the feasibility and efficiency of the proposed approach.

Keywords: Serviced-Oriented Architecture, Software Product Line, Serviced-Oriented Product Line

1 Introduction

The concept of Software Product Line (SPL) is introduced to software engineering to meet requirements of software development in large-scale customization environment. Traditionally, SPLs are implemented with systematically developed components [1]. However, the reconfigurations of these components are largely limited to design-time. Due to the continuously change of user needs and expectations, we need the Service-Oriented Product Line (SOPL) to support the ability of changing products at runtime, by combine the service-oriented Architecture (SOA) and SPL [2-6]. In a SOPL, the service characteristics, e.g., dynamic discoverability and binding [7], can be used to provide the flexibility of reconfiguration for SPLs and their products.

First, how to identify and design services or service compositions for the domain, to decide the variation points and service variability implementation mechanisms, and to define the SOPL architectural view [8-9]. Second, how to provide a common customization framework for different SOPLs to achieve reusability at the level of SPL. Third, using services in SPLs increases the complexity of application derivation, since each function of SOPL is implemented by a service. And several services may implement a single functionality, or more than one functionality can be implemented by a single service. Thus how to select the appropriate services to derive an optimal application that may be composed of several services is also challenging.

Software architecture design plays a fundamental role in coping with inherent difficulties of the development of large-scale and complex software [10]. In the SPL context, reference architecture specifies a common structure for all member products, while application architecture, which can be derived from reference architecture, refers to the concrete architecture of a specific product. Reference architecture can be used as a basis for the development of systems for a specific domain, and application architecture can be used to derive product instances by guiding how to organize the components or services to form the required products. Thus, to solve above challenges, a systematic architecture-centric approach for SOPL development is proposed in this paper.

Since the de facto standard Business Process Execution Language (BPEL) is used so widely in industry that we regard the SOPL as a web service composition system based on it in this paper.

First, the BPEL-Based architecture style and a

*Corresponding Author: Gaoqi He; E-mail: hegaoqi@sei.ecnu.edu.cn
DOI: 10.3966/160792642019072004001

description language bpel4Arch are developed as the basis of the proposed approach, in order to represent and document the architecture asset, and to facilitate the evolution of the proposed approach. Then, we propose an architecture-centric development approach for SOPL, including model-driven reference architecture generating, rule engine based application architecture customization, optimal partition of application architecture. At last, an initial case study on E-Shopping domain, the optimal partition performance evaluation and derivation efficiency evaluation show the feasibility and efficiency of the proposed approach.

The remainder of this paper is organized as follows: Section 2 represents the related work on SOPL. We give an overview of the proposed architecture-centric approach for SOPL development in section 3. Section 4 describes more details of the proposed development approach for SOPL. We evaluate the performance of the proposed approach for SOPL in section 5. Finally, our concluding remarks are provided in section 6.

2 Related Work

Many works [3-5] in the literature has considered the development of SOPL. However, most of them mainly focus on identifying services and variability management in the domain engineering, while few papers refer to complete methodological approach from requirement to implementation. In this paper, we focus on complete methodological approach for SOPL. The proposed approach is similar to [11], where the authors define a Business Process Modeling Notation (BPMN) and a feature model, and then they use feature selections to generate a concrete BPMN model, according to the mapping between features and BPMN model [12-13]. Different from [11], we use the SPL architecture to organize services and guide product derivation process. Architecture has more powerful description ability, especially for complex systems that can't be represented by BPMN. In addition, the mapping between features and services in [11] is simple one-to-one, but we deal with more complex mapping between features and services, and provide an optimal partition for application architecture to derive optimal product instances.

Software Architecture, which depicts the system configuration states from a global perspective, can be effectively used as the basis of software development. Some initiatives have suggested exploring these architectures in SPLs [14-16]. To easy the SPL development and automate the SPL architecture design, [14] introduces a multi-objective optimization approach to SPL architecture design by using evolutionary algorithms. For giving more details on how and when to use reference architectures, [16] explores the use of reference architecture in the development of product line artifacts, and [15] presents an architecture-centric approach to derive product instances from customization of product line architecture. Compared with these works, our approach focuses more on SOA paradigm and describes in more detail Architecture-Centric development process for SOPL, from the views of reference architecture and application architecture.

In addition, some works concerning the generating of architecture in SOPL are proposed in [17-18]. In [17], an approach to designing SOPL architecture for business process families has been proposed. [18] also focuses on approach of architecture designing in SOPL, and provides a detailed description of activities for reference architecture generating. However, both of them pay less attention on model-driven methods and architecture customization in application engineering. Our method towards SOPL architecture not only pays more attention on model-driven development for reference architecture but also provides a common architecture customization approach based on the rule engine. [19] developed a model-driven product derivation tool, called GenArch, centered on definitions of feature, architecture and configuration models, which enable the automatic instantiation of SPLs. However, this tool aims to derive a product based on components. There are no details how to get the initial versions of these three models and the mapping between features and components is simple. Our method concentrates on the development of service-based product lines. We also provide an optimal partition for complex mapping between features and services to derive optimal product instances. And the evaluation shows that our method has higher efficiency on product derivation and modification than the tool GenArch.

3 Approach Overview

The engineering process for the SOPL using the proposed architecture-centric development approach is depicted in Figure 1. Similar to traditional development of SPL, the approach consists of two phases: domain engineering and application engineering. As the phase "develop for reuse", domain engineering is in charge of developing a common reusable infrastructure and assets such as feature model, business process, service library, and reference architecture. While application engineering aims to develop the target products using the assets created before. It is known as the phase "developing reusing". It is during this phase where products are built by service composition based on BPEL.

As the basis of this approach, the BPEL-Based architecture style and an architecture description language bpel4Arch for SOPL are developed first in this paper. Then, the proposed architecture-centric development approach for SOPL pays more attention on following phases: (a) Model-driven reference

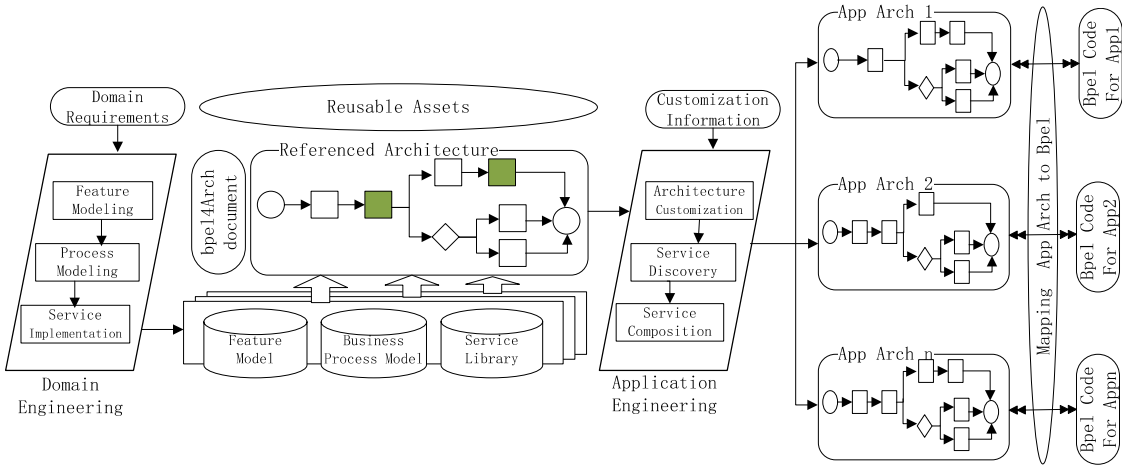


Figure 1. SOPL Development Process Using Our Proposed Approach

architecture generating during domain engineering; (b) Architecture customization based on rule engine during application engineering; and (c) Mapping application architecture to BPEL.

Model-driven reference architecture generating aims to get the reusable reference architecture from feature model and business process models. In this method, we integrate the feature modeling and business process modeling to reference architecture development, for satisfying the service-oriented paradigm and providing demanded variability and flexibility of SPLs. Architecture customization is to derive application architecture based on reference architecture and user's feature selections. In order to simplify this process, we provide a common architecture customization approach for different SPLs based on rule engine. Through this way, we only need to create a rule-based knowledge base for the domain of each SPL and all the tasks of application customization can leave to the common rule engine based customizer. The main task of mapping application architecture to BPEL is to create the mapping relationship between abstract service components of application architecture and concrete service candidates in domain service library, and generate final BPEL code for each customization. To get the BPEL code with minimum number of molecular services, we propose an optimal partition method for application architectures by using the MIP technique.

4 Approach Details

4.1 Architecture Style and Description Language

As described before, each product derived in the application engineering can be regarded as a BPEL-based composite service. Thus a BPEL-based architecture styleis proposed in this paper. As Figure 2 shows, each web service of the SOPL mounts on a BPEL process, which can start and monitor the execution of process. Due to different granularities, a web service can be a molecular service or a BPEL sub-flow with several services. Due to the variability of SPLs, four kinds of business activities are involved in the BPEL flow [17]:

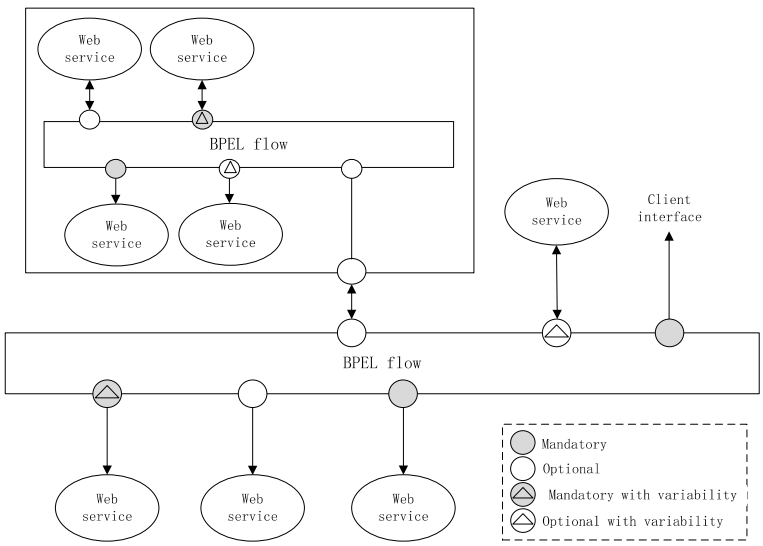


Figure 2. Architecture style for BPEL-based SOPL

- Mandatory business activities with variability: These activities are mandatory, but the variants chosen under the constraints must be decided.
- Optional business process: These activities may or may not be selected in the target products.
- Optional business process with variability: These activities are optional, but the variants chosen under the constraints must be decided.
- Mandatory business activities: These activities must be included in all the products.

According to such architecture style, all the target products derived from application engineering should be translated into BPEL. Thus an architecture description language bpel4Arch is also proposed for SOPL based on BPEL, to formally describe SOPL architecture and facilitate this translation. Two main views need to be described by bpel4Arch: reference architecture and application architecture. Reference architecture includes variability of the domain in SPL and can be used to derive application architecture based on user's customization information. More precisely, Figure 2 describes the reference architecture style. Application architecture style is similar to it just without variable business activities.

Bpel4Arch consists of three parts: service component definition, connector definition and configuration definition. Service component is the entity of services used to describe the activity of business process in SOPL; Connector which is used to create interaction between service components, represents howthe BPEL process controls interactions between web services; Configuration defines the instances of service components and connectors, and creates the relationships, which equal to the bindings between web services in BPEL process. For simplicity, we don't develop all the parts from scratch, but extend some syntaxes of BPEL. These extensions can be divided into two categories: (1) removing the unused tags and annotations that with no help to architecture description from the BPEL syntax; (2) adding variability specific syntax elements to the key tags and annotations reserved from the BPEL syntax, to support the description of reference architecture for SOPL. These additions should contain three aspects of variability information: the point at which variation can occur in reference architecture; the variants which can be bound at some variation point; the variability binding properties such as the variability type, cardinality of variants, relationship of variations, and variability binding constraints.

Table 1 lists the main syntax elements of bpel4Arch, as described before, these syntax elements are derived from two ways. Tags such as <sequence>, <flow>, <pick>, <switch>, <while>, <condition>, <otherwise>, <case>, and <invoke>are derived from the BPEL syntax. Tag <serviceComponent>and attributes id, name, type and feature are new added to our syntax to support the variability description of SOPL architecture.

Table 1. Syntax elements of bpel4Arch (D: Derived from BPEL; A: Added new)

Element	Type	Description
<sequence>	D	Similar to BPEL, service components in this structure style are invoked on sequence.
<flow>	D	Similar to BPEL, service components in this structure style can be invokedin parallel.
<pick>	D	Only one service component in this structure will be invoked.
<condition>	D	A condition whose value can be true or false according to its Boolean expression.
<otherwise>	D	A condition whose value is opposite to its Boolean expression.
<case>	D	Similar to the case term in programming language, it presents a possible invoke path.
<switch>	D	Similar to BPEL flow, the first service components in this structure style will be invoked if its condition is fulfilled.
<while>	D	Service components in this structure style will be invoked repeatedly.
<invoke>	D	It identifies a service component will be invoked here.
invokeType	A	Attribute of <invoke>, identifies the type of service component. Its value can be mandatory, optional, mandatory with variability, and optional with variability.
<serviceComponent>	A	Basic units of SOPL architecture, it represents a service component.
<bpel4Arch>	A	It is the root element of a bepl4Arch file.
archType	A	Attribute of <bpel4Arch>, it identifies reference or application architecture.
domain	A	Attribute of <bpel4Arch>, it specifies the domain of SOPL.
feature	A	Attribute of <serviceComponent>, it indentifies the name of feature that is mapped by this service component.
feature_model_uri	A	Attribute of <serviceComponent>, it describes the uriof thefeature model.

4.2 Model-driven Reference Architecture Generating

As one of the most prominent techniques to provide an abstract overview of architecture models, feature modeling is used widely in current SPL methods [20-22, 29]. However, feature modeling mainly focuses on the description of domain features, without providing representations of basic processes in a system's architecture [23-25]. On the other hand, it is becoming

increasingly clear that a SOA cannot be useful without a clear focus on the business processes. Thus, a model-driven approach for reference architecture generating is proposed in this paper, to derive a reference architecture thatsatisfies demanded SPL variability, flexibility and business process logic.

As Figure 3 shows, this approach starts with an identification phase that is separated in component identification activity and service identification activity. Component identification takes an analysis of feature models to identify the architectural component candidates. Service identification aims to derive service candidates from business process models [26]. In this activity, processes themselves, their sub-processes and process activities can be considered as molecular or orchestrating service candidates, depending on their granularities.

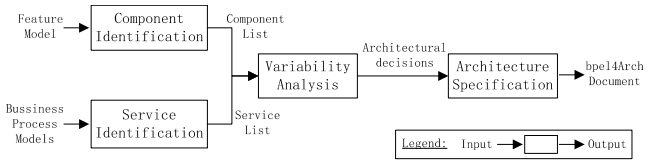


Figure 3. Activities of the Approach for Reference ArchitectureGenerating

The identification phase is followed by a variability analysis activity, which pays more attention on concrete implementation details of variability in the components or services. It receives the list of components and services identified previously to define and document the key architectural decisions regarding variability. Variation in reference architecture can be managed in two ways: variation point description and product line patterns. Service orientation can be used as a technique to implement variability, i.e., each variant can be implemented as a service. It is also necessary to introduce service internal variability through changing a class attribute, a class, a method or even a service. Product line patterns are used to model structural varieties through identifying commonality and variability of related business processes.

Finally, architecture specification activity concludes this approach, in which the components, services, service orchestrations and their flows will be specified. For facilitating the translation between reference architecture and BPEL codes, the architecture description language bpel4Arch is used to describe and document the reference architecture of SOPL, after identifying the basic structural units (components and services) and defining the design decisions (variation decisions, quality decisions, etc.) of the reference architecture.

4.3 Architecture Customization based on Rule Engine

After the reference architecture has been modeled and implemented, feature-driven dynamic customization will be carried out to derive the application architecture for the target product during application engineering. A common architecture customization approach for different SOPLsis proposed based on rule engine. Figure 4 depicts the architecture of this rule engine based customizer.

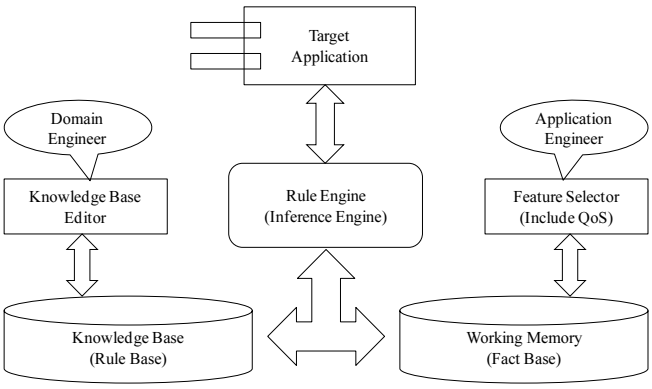


Figure 4. Architecture of Rule Engine Based Customizer

The rule base contains the rules in the form of if-then statements, which represent the knowledge about the domain of SOPL. It is often provided by domain engineers through the knowledge base editor. The working memory contains the facts, which are smallest pieces of information supported by the rule engine. In our customizer, the facts are feature selections and non-functional property constraints provided by application engineers. The rule engine matches facts in the working memory against rules in the rule base, and determines which rules are applicable according to the reasoning method. Thefeature selector can help application engineer select desired features and non-functional profiles. All these information can be used to execute the customization task. The knowledge base editor can help domain engineers develop and edit the domain knowledge for SOPLs.

The feature-driven customization process in above rule engine based customizer is as follows: First, application engineers select desired features and non-functional profiles through the feature selector for each concrete task of customization. After receiving all the customization information, the customizer will add them to the working memory as a set of facts. Then the reasoning task will be executed to check the correctness and consistency based on the feature model and feature dependency rules in the SOPL domain knowledge base, for determining the final features that will be included in the target product. After that, the service components, which are mapped to the final feature selections based on the dependency rules between features and service components, will be reorganized to derive customized application architecture based on the assembly rules extracted from the reference architecture. For more details how to

achieve architecture customization please refer to our publication [27].

4.4 Mapping Application Architecture to BPEL

Through reference architecture generating and architecture customization, we get the application architecture that satisfies user's functionality and QoS requirements. However, each service component appears in this architecture has not been bound to a concrete service, but an abstract service identified by a feature. To get a target product that can be executed on BPEL engine, we still need to translate this architecture to BPEL codes.

For simplicity, traditional methods often map an abstract service node to one component or molecular service. These methods are easy to implement, however, in the real world, the fixed one-to-one mapping cannot satisfy required flexibility and scalability of SOPL. In addition, as one kind of large granularity component, service is being designed to implement more and more functions. So we pay more attention on the many-to-one mapping between abstract service nodes and a molecular service in this section, especially how to partition abstract service nodes, and map them to concrete molecular services to achieve the entire function for target product with minimal number of molecular services.

4.4.1 Hypothesis and Definitions

For simplicity, only linear abstract service node composition model is discussed here. We assume the function of each abstract service node is implemented by at least one molecular service. We first give the following definition of serviceable factor (SF).

Definition 1. Serviceable Factor (SF) identifies the probabilities of that the function of a feature or a feature composition can be implemented by a molecular service. The value varies from 0 to 1. According to our hypothesis described before, SF of each single feature is 1. In the following, we will discuss how to calculate the SF for a feature composition.

Definition 2. Semantic Similarity identifies the degree of similarity between any two features. It focuses on the topology structure of the feature model, without considering the relationships. The semantic similarity between any two features can be calculated as follows:

$$Sim(X,Y)=1,\ where\ X=Y\ or$$

$$Sim(X,Y)=a\times\frac{D_{\max}-Distance(X,Y)}{D_{\max}}+b\times\frac{\left|FeatureSet(X)\cap FeatureSet(Y)\right|}{\left|FeatureSet(X)\cup FeatureSet(Y)\right|}+c\times\frac{L_{\max}-\left|Level(X)-Level(Y)\right|}{L_{\max}},\quad (1)$$

$$where\ X\neq Y$$

a, b, c are weighting parameters, which take on values in the range [0, 1] and reflect the contribution of distance, semantic overlap, and hierarchy level gap on semantic similarity. Their sum is required to be one, but how to set concrete values for them depends on specific feature models, including the largest path length, the number of nodes, and the hierarchy depth of the feature model. Generally their values are set according to domain experts or by experimental attempts to evaluate which combination fits more to the real world. $Distance(X,Y)$ represents the path lengthfrom feature X to feature Y and the lengthof each edge is 1; $D_{\max}$ is the largest distance for all pairs of features in the feature model; $FeatureSet(X)$ represents the feature set from feature X to the root R. $Level(X)$ represents the hierarchy level of feature X in the feature model. $L_{\max}$ is the largest hierarchy level (depth) of the feature model.

Definition 3. Semantic Relevance identifies the relationship of relevance between any two features in the feature model. The semantic similarity reflects the similarity between features on the topology structure of the feature model, however, there are also some kinds of relationships between any two features in the domain of SPL: *Mandatory*, *Optional*, *Or*, *Alternative*, *Require*, *Exclude* [25, 32]. When calculating the semantic relevance, we add an edge between feature X and Y, if X requires Y or Y requires X. For the relationship *Exclude*, there is no edge between the two features. Thus the distance is generally smaller than two, if a pair of features exists some positive (not *Excludes*) relationship. The formula of semantic relevance is as follows:

$$Rel(X,Y)=1\ where\ X=Y$$

$$Rel(X,Y)=\frac{\mu}{Distance(X,Y)+\mu},\ where\ X\neq Y \quad (2)$$

μ is an adjustable parameter. Though μ depends on specific feature model, generally we suggest it takes on value 3, since the distance is generally smaller than two if some positive relationship exists.

Semantic similarity reflects the implication relations while semantic relevance reflects specific relationships defined by domain experts, the two indicators both have contributions to the serviceable factor. We used three different comprehensive analyzing methods, e.g. average value, product value, and the method similar to addition formula of probability, to test which combination of semantic similarity and semantic relevance is more fit to actual cases, i.e. the higher the serviceable factor of each method for the combination

with any two features, the higher the probability to find out a molecular service in the real-world UDDI or service library. After comparing the calculated serviceable factor of each method with service matchingin the real-world UDDI or service library, we find that the third method is more fit to actual cases through the experimental selections. The used formula of serviceable factor is as follows:

$$SF(X,Y) = Sim(X,Y) + Rel(X,Y) - Sim(X,Y) \times Rel(X,Y) \quad (3)$$

Let $F_n = \{f_1, f_2, ...f_n\}$ represent a composite feature set that has n features, and the serviceable factor of F_n can be calculated as follows:

$$SF(F_n) = \frac{\sum_{i \neq j} SF(f_i, f_j)}{C_n^2} \quad i \le n \text{ and } j \le n \quad (4)$$

4.4.2 The Optimal Partition Algorithm

To partition the application architecture for deriving the target product with minimum number of molecular services, enumerating all the possible partitions is necessary. However, the complexity of this kind of method is so high that cannot be applied to practical applications, especially when the number of abstract services is high. Thus, to reduce the number of match between features and all molecular services in UDDI or service library, we propose to use MIP technique to find the optimal partition, where the number of molecular services is minimum and the probability to find a molecular service for each segment is high.

As the first step of this optimal method, we define a matrix $SF[n,n]$ to denote the serviceable factor of each segment of the abstract service component nodes, i.e. $SF[i,j]$ denotes the serviceable factor of the segment of abstract service nodes form node $i+1$ to $j+1$, where $0 \le i < n$, $0 \le j < n$. So for each i and j, the value of $SF[i,j]$ can be calculated according to Eq. (4) if $i < j$. When $i = j$, that means there is only one abstract service node in this segment, so the value of $SF[i,j]$ is 1 according to our hypothesis.

Then we use MIP model to find the optimal partition for the sequential abstract service nodes. First, we define the binary decision variable X_{ij}, $i \le j$ for each abstract service segment from node i to j such that $X_{ij} = 1$ if this abstract service segment is selectedto be included in the partition, and $X_{ij} = 0$ otherwise. Furthermore, in order to reduce the number of variable X_{ij} and improve the efficiency of our MIP model, we set a default value 0 for X_{ij} when $i > j$, and then we execute a first filtering through setting $X_{ij} = 0$ when $SF[i,j] < \tau$, τ is an adjustable parameter, which indentifies the threshold of serviceable factor to find a molecular service. As analyzed before, we can select n abstract service segments at most in the worst case. So we use the following allocation constraints in the model:

The objective function of our MIP model is to minimize the number of molecular services needed to match with the features and maximize the sum of serviceable factor of these abstract service segments. So the objective function can be expressed as follows:

$$\text{minimize}(\alpha \times \frac{\sum_{i=0}^{n-1}\sum_{j=i}^{n-1} X_{ij}}{n} - \beta \frac{\sum_{i=0}^{n-1}\sum_{j=i}^{n-1} (SF[i,j]-\tau) * X_{ij}}{n(1-\tau)}) \quad (5)$$

α and β are contribution factors of the two features: the number of needed molecular service and the sum of serviceable factor.

Abstract service segments in the partition cannot overlap each other, so we should make sure only one abstract service segment which starts from node i or ends to j can be selected at most in the partition. Thus the following constraint should be added:

$$\sum_{i=0}^{a}\sum_{j=a}^{n-1} X_{ij} \le 1, \quad \forall\, 0 \le a \le n-1 \quad (6)$$

Furthermore, the selection of partition must ensure all the abstract service component nodes have been included in. From Eq. (6) we know each abstract service component node can be selected once at most, so we add the following constraint to the model:

$$\sum_{i=0}^{n-1}\sum_{j=i}^{n-1} (j-i+1) X_{ij} = n \quad (7)$$

By solving this model using any MIP solver method, we can get an optimal set of abstract service segments, and then we will test whether this partition can find a molecular service for each abstract service segment. In most of cases, this try will succeed, because we have made the service factor of each candidate abstract service segment is larger than the given threshold and the sum of service factor is as large as possible under other constraints. In the rare cases that the best selection cannot meet the requirements, we can still rapidly get the best one from the rest selections through executing MIP solver again, by adding the following constraints to the model:

$$\sum_{z=0}^{m} X_{ij}^{z} < m+1 \quad (8)$$

Where X_{ij}^{z} is the set of abstract service segments that are selected in the last optimal partition. The purpose of this constraint is to exclude the last optimal partition solution from the rest candidates in the next solving process. Then we will repeat this process until finding a partition that can find a molecular service for each abstract service segment. Compared with the enumerating methods, the number of needed service

matches of our partition method is much smaller and the efficiency is higher. After deriving the optimal partition we can use it to translate application architecture to BPEL for the target product through replacing abstract service segments with selected molecular services. Due to space limitations, the concrete details of these replacements are omitted here.

5 Implementation

5.1 Case Study

To clarify and explain the proposed approach, we introduce an initial case study on the E-Shopping domain in this section. Part of the feature model of the E-Shopping SOPLis presented in Figure 6, by using the protegeto document and analyze feature models [28].

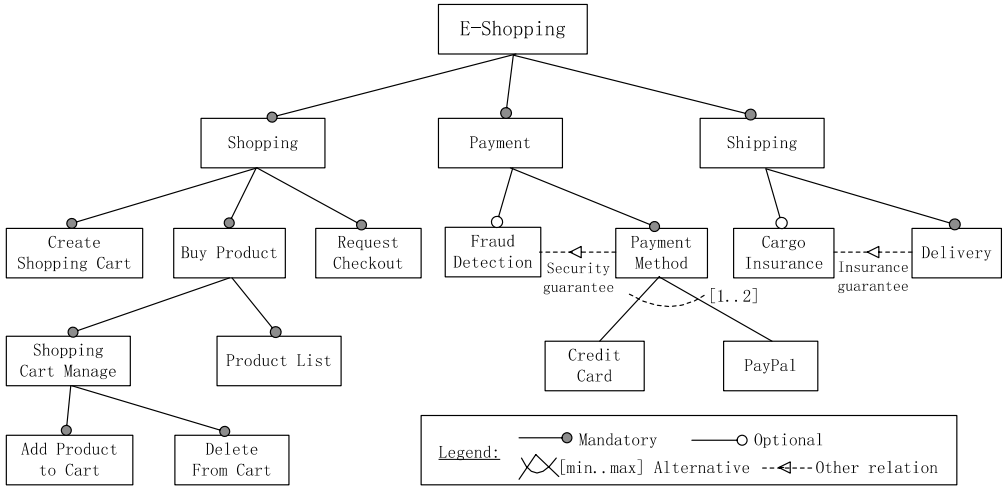


Figure 6. Part of Feature Model of E-Shopping Domain

Applying the technique for component identification, we finish with the following component candidates: create shopping cart, product list, add product to cart, delete from cart, request checkout, fraud detection, credit card payment, PayPal payment, cargo insurance and delivery components. There are also some other components, e.g. access control, customer management that were excluded from this figure due to the space limitation. Figure 7 depicts the simplified business process for the domain of E-Shopping. It starts with an activity that creates a shopping cart. Then customers can get the details of products in the view of product list. After adding products to shopping cart, customers can request a checkout operation. Subsequently, customer will make a decision of selecting cargo insurance or not, before delivering the products. At last, it's time to pay for these products. An optional Fraud Detection will be executed before selecting a concrete payment method that can be credit card or PayPal, in order to support versatile security functionalities in different levels.

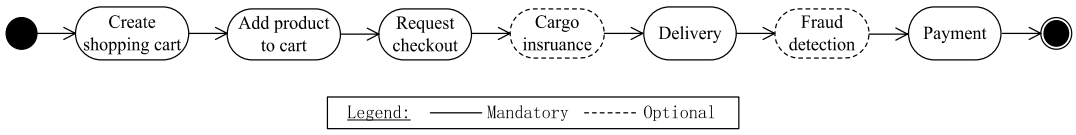


Figure 7. E-Shopping Business Process

From the business process, following service candidates were identified first: create shopping cart, product list view, shopping cart management, request checkout, cargo insurance, delivery, fraud detection, credit card payment and PayPal payment. Then, during variability analysis phase, the cargo insurance and delivery service candidates can be reduced to only one service through introducing variability to the service interface, to reflect the service operation cargo insurance is optional. In the case of payment method feature, we use service orientation technique to implement the variability. Credit card payment and PayPal payment are the two services implementing the variants of payment method feature. The optionalfraud detection feature is also implemented as a service. In addition, due to low variability granularity, sub-features of shopping cart manage (add product to cart and delete from cart) were put all in a unique component with internal variability.

After all architectural decisions derived, the reference architecture (Figure 8) of E-Shopping SOPL is documented by a bpel4Arch file. A code fragment of this file is described in Figure 9. As mentioned before, the bpel4Arch code consists of the Tags of definition and properties of service components, and the Tags of topology relationship between these service

components.

Then we can transform the feature model and reference architecture to rules by applying the technique proposed in our previous work [25], to achieve the customization based on the rule engine. As Figure 10 shows, the feature model and reference architecture are first transformed to the First Order Predicts, e.g. the definition, type, and relationship of features, and the definition, type, and mapping relationship of service components, etc. Then we assume the user selects both the *cargo insurance* and *fraud detection* features with the feature selector, and the selected payment method is *credit card*. The rule engine based customizer will use above predicts and user's selections to derive the application architecture, by using the customization rules. Here we just list three example rules for this case study, the first is to determine the internal variability of service component *Shipping service*; the second is to determine the optional service component *fraud detection*; the third is to determine the alternative service component *credit card payment*. After the customization, we can get the application architecture of E-Shopping SOPL where each variation point in the reference architecture will be bind to a specific variant.

The process view of application architecture of E-Shopping SOPL is depicted in Figure 11. From this figure we can see that there are six sequential activity nodes. Each activity node can be implemented in at least one molecular service according to the assumption. We also assume there is an external molecular service security credit card payment that can achieve the function of Fraud detection and credit card payment service. To get the structure of BPEL with minimum number of molecular services, we use the MIP technique to derive the optimal partition of these activity nodes. Table 2 depicts the calculated serviceable factor matrix.

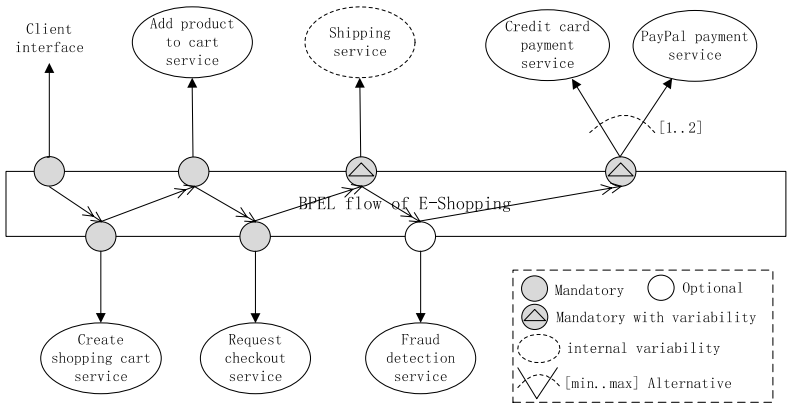


Figure 8. Reference Architecture of E-Shopping

```
<bpel4Arch name="ES_RefArch" archType="reference" domain="E-Shopping"
  feature_model_url="http://www.abc.com/e-shopping-feature-model">
  <serviceComponent name="create_shopping_cart_SC" feature="Create Shopping Cart"/>
  <serviceComponent name="add_product_to_cart_SC" feature="Add Product to Cart"/>
  <serviceComponent name="request_checkout_SC" feature="Request Checkout"/>
  <serviceComponent name="shipping_SC" feature="Shipping"/>
  <serviceComponent name="fraud_detection _SC" feature="Fraud Detection"/>
  <serviceComponent name="credit_card_payment_SC" feature="Credit Card"/>
  <serviceComponent name="paypal_payment_SC" feature="PayPal"/>
  <sequence>
    <invoke name="create_shopping_cart_service"
      serviceComponent="create_shopping_cart_SC" invokeType="mandatory" />
    <invoke name="add_product_to_cart_service"
      serviceComponent="add_product_to_cart_SC" invokeType="mandatory" />
    <invoke name="request_checkout_service"
      serviceComponent="request_checkout_SC" invokeType="mandatory" />
    <invoke name="shipping_service"
      serviceComponent="shipping _SC" invokeType="mandatory with variability" />
    <invoke name="fraud_detection_service"
      serviceComponent="fraud_detection_SC" invokeType="optional" />
    <invoke name="payment_service" invokeType="mandatory with variability">
      <List name="payment_SC_List" type="alternative" min=1 max=2>
        <serviceComponent="credit_card_payment_SC" />
        <serviceComponent="paypal_payment_SC" />
      </List>
    </invoke>
  </sequence>
</bpel4Arch>
```

Figure 9. bpel4arch code fragment for the E-Shopping case study

Feature Predicts:
feature(Shipping),
feature(Cargo Insurance),
feature(Fraud Detection),
feature(Credit Card),
parent_is(Cargo Insurance, Shipping),
type_is(Shipping, mandatory),
type_is(Cargo Insurance, optional),
type_is(Fraud Detection, optional),
type_is(Credit Card, alternative)
User's Selections:
selected(Cargo Insurance, true),
selected(Fraud Detection, true),
selected(Credit Card, true)

Architecture Predicts:
serviceComponent(shipping_SC),
serviceComponent(shipping_with_insurance_SC),
serviceComponent(fraud_detection_SC),
serviceComponent(credit_card_payment_SC)
type_is(shipping_SC, mandatory_with_variability),
type_is(fraut_detection_SC, optional),
type_is(credit_card_payment_SC, optional)
feature_map(shipping_SC, Shipping),
feature_map(shipping_with_insurance_SC, Cargo Insurance),
feature_map(fraud_detection_SC, Fraud Detection),
feature_map(credit_card_payment_SC, Credit Card),
sequence(shipping_SC, fraut_detection_SC),
sequence(fraut_detection_SC, payment_SC_List)

Customization Rules:

(1) feature(Cargo Insurance) ^ parent_is(Cargo Insurance, Shipping) ^ serviceComponent(shipping_SC) ^ feature_map(shipping_SC, Shipping) ^ type_is(shipping_SC, mandatory_with_variability) ^ selected(Cargo Insurance, true) → ArcIncluded(shipping_with_insurance_SC , true) ^ sequence(shipping_with_insurance_SC, fraud_detection_SC)

(2) feature(Fraud Detection) ^ selected(Fraud Detection, true) ^ serviceComponent(fraud_detection_SC) ^ feature_map(fraud_detection_SC, Fraud Detection) → ArcIncluded(fraud_detection_SC, true)

(3) feature(Credit Card) ^ selected(Credit Card, true) ^ serviceComponent(credit_card_SC) ^ feature_map(credit_card_SC, Credit Card) → ArcIncluded(credit_card_SC, true) ^ sequence(fraud_detection_SC, credit_card_SC)

Figure 10. Rule-based customization for application architecture of E-Shopping SOPL

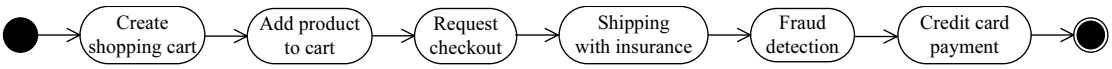


Figure 11. Application Architecture of E-Shopping SOPL

Table 2. Serviceable factor matrix (i: Start index; j: End index)

i \ j	0	1	2	3	4	5
0	1	0.51	0.58	0.55	0.52	0.51
1	0	1	0.51	0.49	0.48	0.49
2	0	0	1	0.57	0.54	0.55
3	0	0	0	1	0.57	0.61
4	0	0	0	0	1	0.80
5	0	0	0	0	0	1

Then, we use the open source lpsolve 5.5 to solve the MIP model setting $\alpha = \beta = 0.5$. From table 2 we can see the serviceable factor for each abstract service segment is between 0.45 and 0.80, and the probability of finding out a concrete service for an abstract service segment is almost zero when the serviceable factor is below 0.7 according to the domain experts, so we set the threshold of service factor $\tau = 0.7$. By running thelpsove once, we derive an optimal partition for this case study. In this partition, abstract service component node 0 to 3 will map a concrete molecular service separately; abstract service 4 and 5 will put together to map a concrete molecular service. Then we will try to find concrete molecular service candidates for each segment of this partition. If we can find all the service candidates, that means this partition is the final optimal partition, else we will run lpsolve again to find the next optimal partition selection through adding the constraints to this MIP model according to Eq. (8).

5.2 Optimal Partition Performance Evaluation

In this subsection, we will evaluate the performance of the proposed MIP based optimal partition method. We first develop a feature mode with more than 200 feature nodes. Then we calculate the value of serviceable factor for any two features according to Eq. (3). Here we set $a = 5,\ b = 1,\ c = 0.2$ for Eq. (1), $\mu = 3$ for Eq. (2), $\alpha = 0.6,\ \beta = 0.4$ for Eq. (5) and $\tau = 0.8$.

Figure 12(a) shows the cumulative distribution function (CDF) of the number of MIP solving with different number of abstract service component nodes *k* separately. The results can be summarized as follows:

- Overall, optimal partition method achieves good results for all examined values of *k*. For $k = 10$, at least 90% of the application architecture can find the optimal partition within one MIP solving. Even

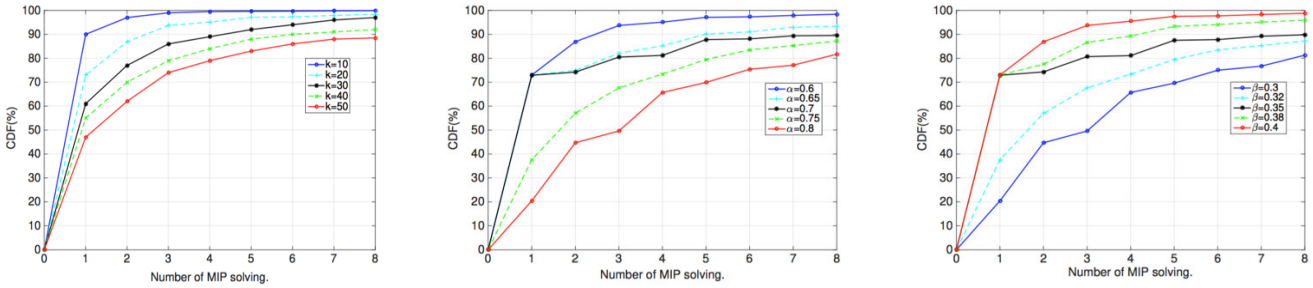
for $k = 50$, at least 80% of the application architecture can find the optimal partition within four MIP solving.

- Larger k achieves lower efficiency. That means if the number of abstract service components is larger, we need more MIP solving to map application architecture to BPEL. Because larger number of abstract service components means more abstract service segments whose serviceable factor is larger than the threshold, so the probability to find molecular services for the partition is lower, resulting in more MIP solving to find the optimal partition.

Figure 13(a) shows the percentage of the number of final selected molecular service under different number of abstract service nodes. We can see that these values are stable near 99%, little affected by the number of abstract service nodes. This distribution shows that our method can achieve stable efficiency on reducing number of final selected molecular service, independent of the number of abstract service component nodes.

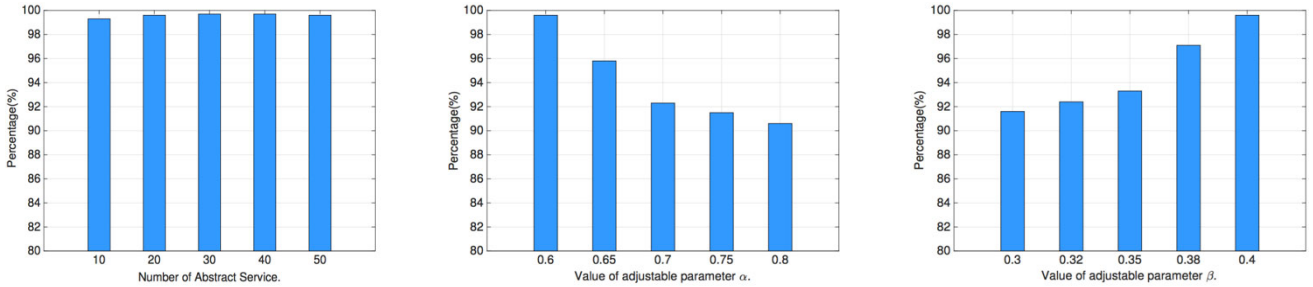
Then we keep the number of abstract service fixed to 20. Figure 12(b) and Figure 13(b) depict CDF of MIP solving and the percentage of selected services under different value of α separately, when $\beta = 0.4$. We can see that larger value of α achieves lower efficiency on number of MIP solving and higher efficiency on number of needed molecular services. The results can be explained as follows: larger value of α means the influence of the number of molecular services on the final selection is strengthened, so in most cases, we need less molecular services to achieve the entire function. However, less molecular services means more abstract service segments with more than one abstract service component node will be tested if can find a molecular service to achieve its function, so we need more number of MIP solving to find the final optimal partition.

On the contrary, as depicted by Figure 12(c) and Figure 13(c) where $\alpha = 0.6$ and β varies from 0.3 to 0.4, larger value of β achieves higher efficiency on the number of MIP resolving and lower efficiency on the number of needed molecular services. The reason is similar to Figure 12(b) and Figure 13(b). Due to space limitation, we don't describe it more detail.



(a) Different number of abstract service component nodes (b) Different valuesof α (c) Different values of β

Figure 12. CDF of MIP solving under



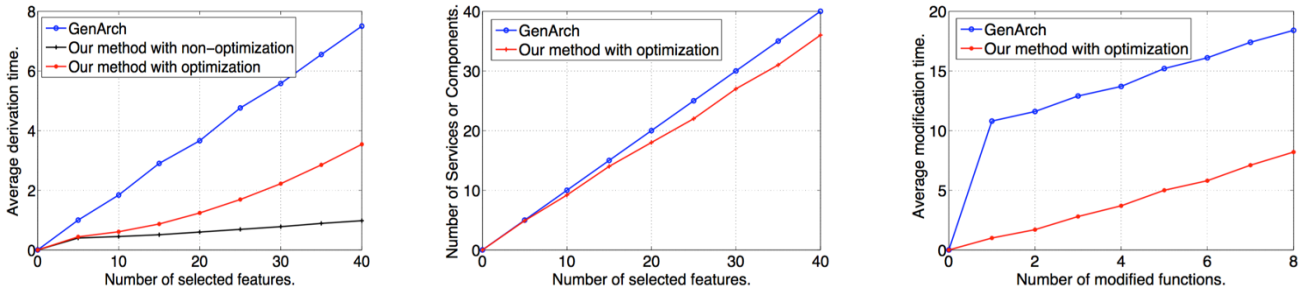
(a) Different number of abstract service component nodes (b) Different values of α (c) Different values of β

Figure 13. Percentage of selected service under

From above analysis, we can see that α and β have greater impact on efficiency of needed molecular services than k. It is thus important to assign appropriate values forthem. However, the optimal α and β depend on specific application scenarios, including user's preference, feature model, and used UDDI. A good rule of thumb is that setting default α and β to 0.5, and then try experimental attempts (with different combinations of them) or use some machine learning method to get the optimal combination of α and β, which can achieve better balance between them and bring highest efficiency on needed molecular services.

5.3 Product Derivation Efficiency Evaluation

Next we report the efficiency results of empirical evaluations of our method on E-Shopping case study, when comparing with GenArch, which is a model-based product derivation tool developed for component-based SPL [24]. For each customization, we use the same feature selections to drive product derivation and record time spent to get the product. Figure 14(a) shows the average derivation time, which is normalized to that spent for customization with 5 selected features by using GenArch, with different number of selected features.



(a) Average derivation time

(b) Required number of services or components for customization with different number of selected features

(c) Average modification time with different number of modified functions

Figure 14.

As we can see, our method achieves 60%-87% efficiency improvement for product derivation without MIP-Based optimal partition. Even with MIP-Based optimal partition, it can also achieve 53%-67% efficiency improvement. The main reason is that GenArch needs to copy the required codes of components to generate a JAVA project, while our method only needs to derive a BPEL file. Figure 14(b) shows the average number of required services or components to derive the target product with different number of selected features. From this figure, we can see that our method achieves better efficiency on number of needed molecular services than GenArch. It is mainly because the optimal partition aims to derive the target products with fewer services by mapping more than one feature to a molecular service.

We also evaluate the average time of our method to change some functions after a product has been derived. We first derive a product with 10 selected features, then we change some functions and record the time to achieve this change. The cost time is normalized to one function modification. Figure 14(c) shows the comparative results on average time to achieve such modifications with different number of modified functions. We can see that our method achieves better efficiency on modification time than GenArch. Because it just needs to modify the implementation details of the required services, while GenArch has to re-execute the derivation process, which is time-consuming, e.g. its average derivation time is nearly 10 times as the time to change one function.

6 Conclusion

In this paper, an efficient architecture-centric approach for SOPL development is proposed. To achieve this development, the BPEL-Based architecture style and the architecture description language bpel4Arch are developed first. Then, based on them, a model-driven reference architecture generating method and a common architecture customization approach are proposed based on the rule engine. Furthermore, for facilitating the translation between application architecture and BPEL codes, we also provide an optimal partition for application architecture according to the mapping between abstract service component nodes and concrete service candidates in the domain service library, in order to achieve the entire function of application architecture with minimum molecular services. Finally, a case study and some evaluations prove the feasibility and high efficiency of the proposed approach. Due to limitation of the number of available services in production, the optimization performance and

derivation efficiency of our method are currently validated only by controlled experiments. However, in the future, we plan to apply and improve the proposed method in the real world SOPL application development.

Acknowledgments

This work was partially supported by the NSF of China under grant No. 61602175, Natural Science Foundation of Shanghai under grant No. 19ZR1415800, the National Science and Technology Supporting Program of China under grant No. 2015BAH18F02, the Model Information Service Industry Program of Guangdong Province under grunt No. GDEID2010IS049, the Fundamental Research Funds for the Central Universities under grant No. ZH1726108, and the Special Funds for Informatization Development in Shanghai under grant No. 201602008, Popularization of Science Program of Shanghai Science and Technology Commission under grant No. 19DZ2301100.

References

[1] B. Mohabbati, M. Hatala, D. Gašević, M. Asadi, M. Bošković, Development and Configuration of Service-oriented Systems Families, *ACM Symposium on Applied Computing*, Taichung, Taiwan, 2011, pp. 1606-1613.

[2] R. Capilla, J. Bosch, P. Trinidad, A. Ruiz-Cortésc, M. Hinchey, An Overview of Dynamic Software Product Line Architectures and Techniques: Observations from Research and Industry, *Journal of Systems and Software*, Vol. 91, No. 1, pp. 3-23, May, 2014.

[3] S. Khoshnevis, An Approach to Variability Management in Service-oriented Product Lines, *International Conference on Software Engineering*, Zurich, Switzerland, 2012, pp. 1483-1486.

[4] J. R. F. Da Silva, A. S. De Melo Filho, V. C. Garcia, Toward a Qos Based Run-time Reconfiguration in Service-oriented Dynamic Software Product Lines, *International Conference on Enterprise Information Systems*, Lisbon, Portugal, 2014, pp. 460-465.

[5] A. Murguzur, R. Capilla, S. Trujillo, Ó. Ortiz, R. E. Lopez-Herrejon, Context Variability Modeling *for Runtime Configuration of Service-based Dynamic Software Product Lines*, SPLC, Florence, Italy, 2014, pp. 2-9.

[6] T. K. Chang, E. T. Y. Hwang, An Soa-based System Modeling Methodology, *Journal of Internet Technology*, Vol. 16, No. 3, pp. 547-561, May, 2015.

[7] C. Yu, D. Yao, X. Li, L.T. Yang, N. Xiong, H. Jin, Location-aware Private service Discovery in Pervasive Computing Environment, *Information Sciences*, Vol. 230, No. 5, pp. 78-93, May, 2013.

[8] M. Abu-Matar, H. Gomaa, An Automated Framework for Variability Management of Service-oriented Software Product Lines, *Seventh International Symposium on Service-Oriented System Engineering*, Washington DC, USA, 2013, pp. 260-267.

[9] S. Khoshnevis, F. Shams, Linear Evolution of Domain Architecture in Service-oriented Software Product Lines, *Fundamentals of Software Engineering*, Tehran, Iran, 2015, pp. 275-291.

[10] B. Tekinerdogan, S. Cetin, M. A. Babar, P. Lago, J. Mäkiö, Architecting in Global Software Engineering, *ACM SIGSOFT Software Engineering Notes*, Vol. 37, No. 1, pp. 1-7, Jan., 2012.

[11] G. H. Alferez, V. Pelechano, Systematic Reuse of Web Services through Software Product Line Engineering, *European Conference on Web Services*, Bertinoro, Italy, 2011, pp. 192-199.

[12] M. Terenciani, D. Paiva, G. Landre, M. I. Cagnin, BPMN- A Notation for Representation of Variability in Business Process towards Supporting Business Process Line Modeling, *International Conference on Software Engineering and Knowledge Engineering*, Pittsburgh, USA, 2015, pp. 227-230.

[13] P. Bollen, Business Process Model Semantics in Bpmn, *Innovations in Enterprise Information Systems Management and Engineering*, Hagenberg, Austria, 2016, pp. 31-45.

[14] T. E. Colanzi, Search Based Design of Software Product Lines Architectures, *International Conference on Software Engineering*, Zurich, Switzerland, 2012, pp. 1507-1510.

[15] C. Cu, Y. Zheng, Architecture-centric Derivation of Products in a Software Product Line, *International Workshop on Modeling in Software Engineering*, New York, USA, 2016, pp. 27-33.

[16] E. Y. Nakagawa, P. O. Antonino, M. Becker, *Exploring the Use of Reference Architectures in the Development of Product Line Artifacts*, SPLC, Munich, Germany, 2011, pp. 1-8.

[17] E. Ye, M. Moon, Y. Kim, K. Yeom, An Approach to Designing Service-oriented Product-line Architecture for Business Process Families, *International Conference on Advanced Communication Technology*, Okamoto, Japan, 2007, pp. 999-1002.

[18] F. M. Medeiros, E. S. Almeida, S. R. de Lemos Meira, Towards an Approach for Service-oriented Product Line Architectures, *Proceedings of the Workshop on Service-oriented Architectures and Software Product Lines*, Jeju Island, South Korea, 2009, pp. 1-7.

[19] E. Cirilo, U. Kulesza, C. J. P. de Lucena, GenArch: A Model-based Product Derivation Tool, *Proceedings of the First Brazilian Symposium on Components*, Architecture and Reuse, Campinas, Brazil, 2007, pp. 31-44.

[20] M. Acher, P. Collet, P. Lahire, R. B. France, Familiar: A Domain-Specific Language for Large Scale Management of Feature Models, *Science of Computer Programming*, Vol. 78, No. 6, pp. 657-681, June, 2013.

[21] N. I. Altintas, S. Cetin, A. H. Dogru, H. Oguztuzun, Modeling Product Line Software Assets Using Domain-specific Kits, *IEEE Transactions on Software Engineering*, Vol. 38, No. 6, pp. 1376-1402, Nov., 2012.

[22] N. Itzik, I. Reinhartz-Berger, *Sova- A Yool for Semantic and*

Ontological Variability Analysis, CAiSE, Thessaloniki, Greece, 2014, pp. 177-184.

[23] I. Reinhartz-Berger, A. Zamansky, Y. Wand, Taming Software Variability: Ontological Foundations of Variability Mechanisms, *International Conference on Conceptual Computing*, Stockholm, Sweden, 2015, pp. 399-406.

[24] H. A. Duran-Limon, C. A. Garcia-Rios, F. E. Castillo-Barrera, R. Capilla, An Ontology-based Product Architecture Derivation Approach, *IEEE Transactions on Software Engineering*, Vol. 41, No. 12, pp. 1153-1168, December, 2015.

[25] Y. Yin, H. Gao, D. Yu, Model-based Data-Intensive Service Abstraction Refinement, *Journal of Internet Technology*, Vol. 14, No. 5, pp. 807-816, September, 2013.

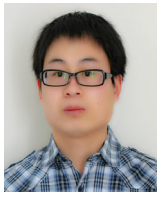
[26] X. Lu, J. Yin, Y. Yin, S. Deng, S. Luo, *Knowledge base-Centric Customization Approach for Different Software Product Lines*, SEA, Marina del Rey, USA, 2010, pp. 427-434.

[27] Y. H. Chang, B. K. Chen, The Systematic Construction of a Valid Domain Ontology, *Journal of Internet Technology*, Vol. 16, No. 2, pp. 289-299, March, 2015.

[28] G. Becan, M. Acher, B. Baudry, S. B. Nasr, Breathing Ontological Knowledge into Feature Model Synthesis: An Empirical Study, *Empirical Software Engineering*, Vol. 21, No. 4, pp. 1794-1841, August, 2016.

[29] N. Xiong, A. V. Vasilakos, L. T. Yang, L. Song, Y. Pan, R. Kannan, Y. Li, Comparative Analysis of Quality of Service and Memory Usage for Adaptive Failure Detectors in Healthcare Systems, *IEEE Journal on Selected Areas in Communications*, 27(4), 495-509, 2009.

Biographies



Xingjian Lu is an assistant professor in the School of Information Science and Engineering, East China University of Science and Technology, Shanghai, China. He received his Ph.D. in Zhejiang University, China in 2014. His research interests include cloud computing, service computing, and software product lines.

Jianwei Yin is a full professor in the College of Computer Science and Technology, Zhejiang University, China. He received his Ph.D. in Zhejiang University, China in 2001. His current research interests include cloud computing, and service computing. He has published more than 120 research papers in these areas.

Gaoqi He is currently an associate Professor at Department of Computer Science and Engineering at East China University of Science and Technology. He received his Ph.D. degree from State Key Laboratory of CAD&CG in Zhejiang University in 2007. His research interests include algorithm design and optimization, and computer graphics.

Huiqun Yu is currently a Professor of computer science with the Department of Computer Science and Engineering at East China University of Science and Technology. He received his Ph.D. degree from Shanghai Jiaotong University in 1995. His research interests include software engineering, cloud computing and formal methods.

Neal N. Xiong is currently an Associate Professor (3rd year) at Department of Mathematics and Computer Science, Northeastern State University, OK, USA. His research interests include Cloud Computing, Parallel and Distributed Computing, Networks, and Optimization Theory. He is serving as an Editor-in-Chief, Associate editor or Editor member for over 10 international journals (including Associate Editor for IEEE Tran. on Systems, Man & Cybernetics: Systems, Information Sciences, and Journal of Internet Technology, Editor-in-Chief of Journal of Parallel & Cloud Computing).