Linear Evolution of Domain Architecture in Service-Oriented Software Product Lines

Sedigheh Khoshnevis, Fereidoon Shams

Linear Evolution of Domain Architecture in Service-Oriented Software Product Lines

Sedigheh Khoshnevis✉, Fereidoon Shams

Department of Computer Engineering, Shahid Beheshti University, G.C., Tehran, Iran
{s_khoshnevis,f_shams}@sbu.ac.ir

Abstract. In service-oriented software product lines, when a change occurs in the business process variability model, designing the domain architecture from scratch imposes costs of re-architecture, high costs of change in the domain-level assets; and high costs of change in many products based on the new domain architecture. In this paper, focusing on the linear evolution scenario in service-oriented product lines, which refers to propagating changes in some assets to some other assets, both in the domain level, we deal with the problem of propagating changes in domain requirements (the business process variability model) to the domain architecture level, in a cost-optimal and consistency-preserving way. We present a method to suggest the optimal change propagation options to reach the aforementioned goals. The method showed promising to provide minimal change costs as well as to fully preserve consistency of the target models if no human intervention exists in the change propagation results.
Keywords: variability. service-oriented product line. linear evolution. Pareto optimization. consistency.

1 Introduction

A software product line is a collection of software-intensive systems that share common properties and are developed aiming at meeting specific market needs or special missions based on a set of core assets [1]. Indeed, a software product line contains a family of software systems that share a number of common functionalities and a number of variable ones. To address the common and variable functionalities, reusable core assets (such as requirements, design artifacts, components, test cases, etc.) are developed, which can be reused by different members of the family[2].

There are two main lifecycles in software product line engineering: domain engineering and application engineering[3]. In the domain engineering lifecycle, the main emphasis is on defining the commonalities and the differences, or in other words determining variability among the aimed products, based on which, the reusable artifacts (the core assets) are developed or obtained. Meanwhile, in application engineering, the core assets are reused in developing particular products of the product line, using the common (mandatory) product elements

and selecting optional and variant product elements, both determined in domain engineering lifecycle.

In a service-oriented software product line, the software architecture building blocks are services, which are identified and specified based on the business processes[4]. The business processes are continually changing and hence, a domain engineer may decide to reflect the changes of the business processes to other artifacts in the software product line. Such a change, therefore, can widely affect artifacts both in the domain and application engineering lifecycles [3].

In general, changing or evolution of a software product line can take place in terms of one or a combination of six *evolution scenarios*, namely, linear evolution, synchronization, merge, propagation, cloning and derivation [5] (see Table 1). These evolution scenarios are different in the *change source* and *change target*. For instance, in the linear evolution scenario, the change source and the change target are both in the domain-level.

Table 1: SPL evolution scenarios [5]

Scenario	Change Source	Change Target	Description
Linear Evolution	Domain Variability Models	Domain Variability Models	-
Synchronization	Domain Variability Models	Product Configurations	-
Merge	Product Configurations	Domain Variability Models	-
Propagation	Product Configurations	Other Product Configurations	-
Cloning	-	New Product Configuration	Based on current product configurations
Derivation	-	New Product Configuration	Based on domain variability models

The scope of research in this paper includes the linear evolution scenario, to propagate changes in a service-oriented software product line, from the requirements-level variability model (which is the business process variability model) to the architecture-level variability model (which mainly includes the variability of services). One way to achieve the mentioned propagation is re-architecting the target architecture model back from scratch based on the changed domain requirements. However, using this way will impose costs, not only for the re-architecture itself, but also for providing new single services and service compositions that may be too distant from their previous versions, and thus will result to: (A) wide changes in the core assets in the domain level; and hence, (B) wide changes in the products that are built by reusing them.

On this basis, in this research we aim to: *"propose a semi-automatic method for linear evolution of service-oriented architecture-level variability model due to the changes in the business process variability model in a way that the costs of change are minimized and that the consistency of the changed model is preserved"*.

To reach this goal, we were inspired by the MAPE-K control loop which is widely adopted in software adaptation in response to changes [6] and we made use of multi-objective Pareto optimization mechanisms to minimize change costs.

Decision making for applying the changes may depend on other factors (such as economic conditions, state of the market, etc.) that cannot be comprehensively determined and formulated in terms of computerized processes; therefore, although it is desirable that the change propagation process be fully automated, we emphasize on a semi-automatic method to avoid missing the role of human

factors and expertise in the change decisions. We expect that the method minimizes the change costs and preserves the consistency of the target model, keeping it free from logical conflicts.

1.1 Motivating Example

As a simple example, suppose that a service-oriented Office Letter Submission (OLS) product line is established by a company. For such a product line, a business process family model is designed, based on which, a service variability model was carried out. Fig. 1 depicts the business process variability model in terms of a Business Process Family Model (BPFM), which is a notation proposed by Moon and her team [7, 8]. It is an extended UML activity diagram augmented with variability-specific elements, such as variation points, optional versus mandatory activities, variant regions (set of variant activities that generally form an optional sub-process), variability relationships (either excludes or requires constraint), etc.

Moreover, we consider the service variability model (which we name SVM for ease) as a set of services and their interfaces (including operations and messages), along with their variability attributes (optional versus mandatory services, operations and messages) and variability relationships ("excludes", "requires", and "or" relationships between services and between operations). An SVM for the sample OLS product line is depicted in Fig. 3.

Suppose that the models represented in Fig. 1 and 3 both belong to the current version (version i) of the sample OLS product line. Note that, for convenience, we have only included the identifiers of activities that are realized by the corresponding service operations in the services; for example in Fig. 3, optional service "Serv1" includes an optional operation "Oper1" that realizes activity "Create Draft for Internal Letters". It has a set of incoming and outgoing messages that are not depicted in Fig. 3, as they are not necessary for the purposes of the current running example.

Let us suppose that the BPFM is requested to change to version *i+1* as depicted in Fig. 2. As it is obvious, the following items of change have occurred in BPFM version *i+1* regarding its previous version.

- Mandatory activity (4) is removed.
- Mandatory activity (7) is changed to optional.
- Mandatory activity (9) is added.

Considering specifically the addition of the new activity (9), we have five change options in the SVM: adding the new operation to: "Serv1", "Serv2", "Serv3", "Serv4", or a new empty service "Serv5". If, for example, for the abstract architecture-level service "Serv1" two concrete services CS1 and CS2 are developed, and if there are 20 configurations in which, CS1 or CS2 are invoked, then a change in "Serv1" propagates not only to those two concrete services, but also to 20 products. Similarly, we would have a number of concrete services for other abstract services in the SVM, each of which could be invoked in a

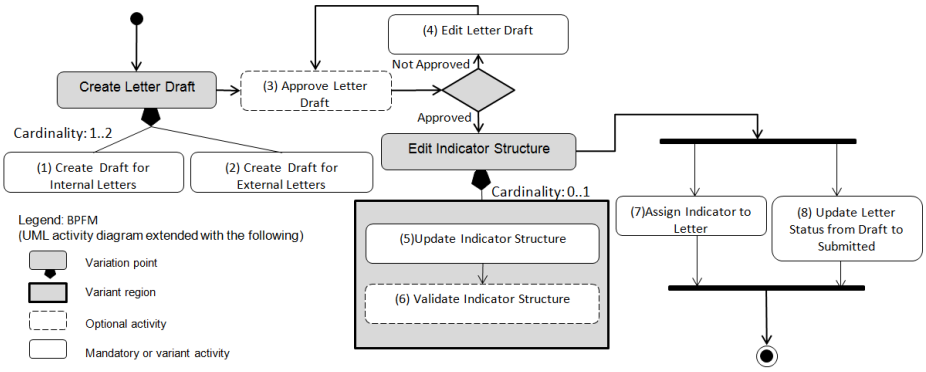

Fig. 1: Current version (i) of the BPFM diagram for the sample Office Letter Submission (OLS) product line

number of product configurations. Therefore, it is crucial to find an optimal set of changes to the SVM so that the change cost, in terms of number of affected artifacts, is minimized.

Moreover, a method to deal with this problem is expected to preserve consistency in the new version of SVM. An inconsistent SVM contains internal logical conflicts between its elements. For example, if there is an "excludes" relationship between two mandatory services, then it is not consistent, since being mandatory implies that in every product configuration, both services are invoked; and the "excludes" relationship implies that only one of them can be invoked in a product configuration, which is contradictory. In another work in progress, we have derived and articulated inconsistency conditions to be checked automatically on SVM models.

In short, the research problem is how to find the set of proper changes in the SVM among a set of change options, so that the cost of change is minimized and the consistency of the target model is preserved. The rest of the paper is organized as follows: In section 2, we take a glance at the previous work on this subject; afterwards, in section 3 we fully describe the proposed method. Section 4 is dedicated to evaluation of the research, while section 5 concludes the paper and proposes a few trends for the future work.

2 State of the Art

The previous work regarding the subject of this research can be divided into two main categories: research works related to software product line evolution, and the research reports on evolution of variability in services and the service-oriented product lines.

Irrespective of service-orientation, there are a number of research works reported for managing and controlling evolution in software product lines (refer to

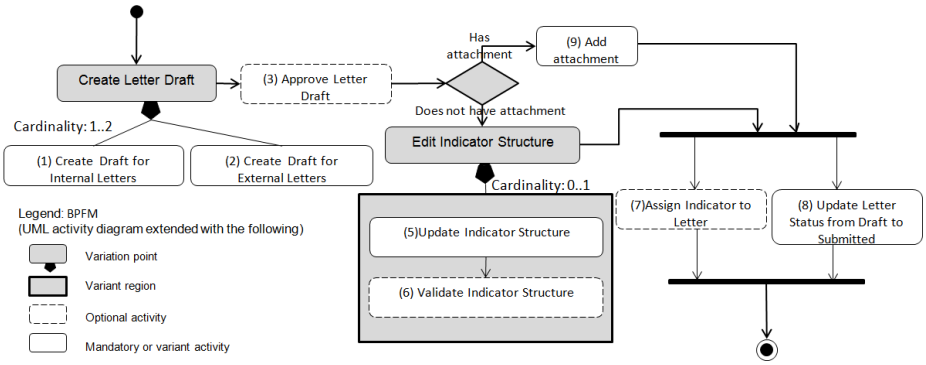

Fig. 2: New version $(i + 1)$ of the BPFM diagram for the sample Office Letter Submission (OLS) product line

[9] for a wide survey in this regard). The common notion among most of these works is two-fold: emphasis on minimizing the cost of change [10–12], and emphasis on consistency preservation in the product line artifacts which are subject to change [10–13].

Another common point of focus among these works is the steps of encountering with a change request. For instance, Ajila and Kaba in [11] propose evolution management mechanisms that fit in a four-phased process, which includes change identification, change impact analysis, change propagation , and change validation. The same has been proposed in service evolution (irrespective of the product line engineering) [14]; for example for services self-adaptation, the well-known MAPE-K loop has been suggested to Monitor, Analyze, Plan and Execute changes, providing and consuming some change Knowledge [6]. However, all of the research works in this first category are feature-model-based; while in a service-oriented product line, we need to support the business processes; thus these methods cannot fit to our problem.

The research works reported on evolving software with respect to the variability of services within or out of the scopes of a product line, fall into two subcategories: (A) Reports on using the concept of variability to support a set of pre-determined changes in single service-oriented software systems (such as [15] and [16]), none of which are in the scope of a software product line, and thus the works in this subcategory does not fit to our problem.

(B) Reports on supporting some type of evolution in service-oriented product lines (such as [12] and [17]); however, in this subcategory, although the research works are within the scopes of service-oriented product lines, there are no emphasis on the business processes as the main source of continuous change. Gomaa and Hashimoto [17] propose a general service-oriented variability model which uses feature models as the main requirements-level variability model, and do not focus on business process variability model. They concentrate on the basics of self-adaptation in service-oriented product lines. Moreover, Hinchey et al.

[12] discuss general issues in dynamic product lines and do not specifically address evolution scenarios in the service-oriented product lines emerging from the changes in the business processes. Hence, the research works in this subcategory, either, cannot be leveraged to solve the problem mentioned in this paper.

3 The Proposed Method

Considering the linear evolution scenario, the source of changes in the domain architecture of a service-oriented product line is the business process variability model. In this research, we specifically chose the Business Process Family Model (BPFM) notation [7, 8] for the purpose of specifying the business process variability. Meanwhile, the change target is the service variability model (SVM). SVM is basically a set of mandatory and/or optional services, each of which contains a nonempty set of mandatory and/or optional operations. Among services and among operations, there can be variability relationships, including, "or", "excludes", and "requires" relationships. Thus, if we consider the proposed method as a black box, then the input to it will be a BPFM′ (the changed version of the current BPFM), and the output from it will be an SVM′ (the changed version of the current SVM). Inside the black box, we embed four main units and a knowledge repository, which support four main operations by sharing some knowledge among them. These four main operations are: change identification, change impact analysis, change planning and change execution, which can continue over to the change identification operation if the result of change execution is exposed to some new change. Fig. 4 depicts these four main operations besides the knowledge repository in which, all of the changes, their details, and rationales are stored. This paper focuses mostly on the first two operations of the proposed cycle.

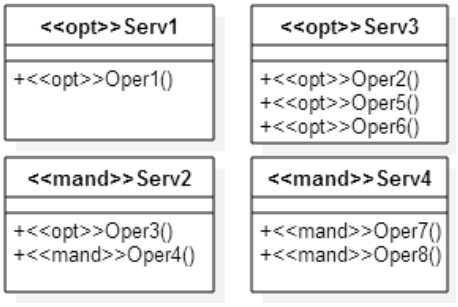

Fig. 3: Current version (i) of the SVM diagram for the sample Office Letter Submission (OLS) product line

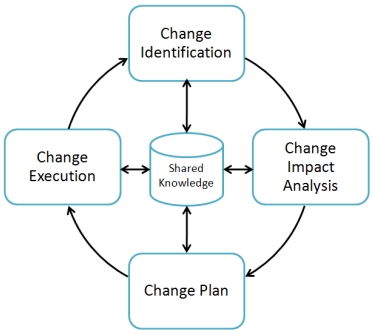

Fig. 4: The proposed change cycle for linear evolution in service-oriented product line

3.1 Change Identification

"Change identification" operation includes the identification of the elements in the source model (BPFM) that are added, removed, or modified. The result, which we represent by ΔB, is in fact, the set of differences of BPFM$'$ (the new version) with BPFM (the old version). Each change item in ΔB, can be an element (e) or a variability relationship (r) between the elements in the BPFM, where each element e (such as an activity, a variation point, or a variants region) is a member of all BPFM elements (E); and each relationship r (such as a control flow, a connection or a constraint) is a member of all BPFM relationships (R). Elements can be added, removed or modified, while relationships can only be added or removed. On this basis we will have formula (1), in which *add* means adding an element or a relationship, *rem* means removing an element or a relationship, and *mod* denotes modifying an element.

$$ChangeItem::= add\ e \mid mod\ e \mid rem\ e \mid add\ r \mid rem\ r \tag{1}$$

Elements and relationships each have their quadruple structures specified in formulas (2) and (3). Formula (2) states that each element has a type (e.g., it is an activity, a variation point, or a variants region), a name, an attribute type, and an attribute value; while formula (3) denotes that a relationship has a category, a type, a source and a target (see [7, 8] for details of BPFM).

$$e = (etype, ename, eattrtype, eattrvalue) \tag{2}$$

$$r = (rcat, rtype, rsource, rdestin) \tag{3}$$

In the example in subsection 1.1, we depicted a list of change items that occurred to the OLS BPFM.

3.2 Change Impact Analysis

The goal of "change impact analysis" operation is to (1) determine all *change options* for reflecting the change items of BPFM to SVM; (2) calculating the cost of applying each change option; and (3) providing a set of optimal change options for the architects and the domain engineers to help them in decision making about which change options to apply.

Determining SVM Change Options: To designate the changes to SVM (ΔS) as a result of changes in BPFM, we should determine all potential change items that can take place in the SVM (we call them change options) in reflection to each change item existing in ΔB; and then we should analyze costs of each change option. Let CI be the set of all change items residing in ΔB, and CO_i be the set of all possible change options in SVM in response to a change item CI_i. As an example, if currently there exists three services $S1$ to $S3$ in the SVM, and some CI_1 corresponds to adding a mandatory activity "a" to the BPFM, then each of the following four items makes up a change option in the set CO_i:

adding a new operation corresponding to activity "a" to (1) $S1$, (2) $S2$, (3) $S3$, and (4) a new empty service $S4$. Then a set CO is formed from the union of all change options co_j in each CO_i. Fig. 5 depicts the relationship between sets CI and CO. For our running OLS example in subsection 1.1, the set of change items and change options are depicted in Fig. 6.

Change Items in BPFM (CI)	CI_1			CI_2		CI_3			...
Change Options in SVM (CO)	co_1	co_2	co_3	co_4	co_5	co_6	co_7	co_8	...
Selection Vector	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	...

Fig. 5: The relationship between change items in BPFM and change options in SVM

Change Items (CI)	Remove activity (4)	Change activity (7) from mandatory to optional	Add new activity (9)				
Change Options (CO)	Apply	Apply	Add to serv1	Add to serv2	Add to serv3	Add to serv4	Add to new serv5
Selection Vector index	v_1	v_2	v_3	v_4	v_5	v_6	v_7
Solution $z1$	1	1	1	0	0	0	0
Solution $z2$	1	1	0	1	0	0	0
Solution $z3$	1	1	0	0	1	0	0
Solution $z4$	1	1	0	0	0	1	0
Solution $z5$	1	1	0	0	0	0	1
$c1$	1	0	1	1	1	1	1
$c2$	1	0	1	1	1	1	1
$c3$	1	0	0	0	0	0	0
Affected Services	$S2$	-	$S1$	$S2$	$S3$	$S4$	New Empty $S5$

Fig. 6: Potential solutions for the OLS sample

Moreover, to build up a single solution (a potential ΔS) we need to select one of the change options for each change item. The selection is done by designating a Boolean selection vector as shown in Fig. 5, with the constraint that for each change item CI_i one and only one of the corresponding change options should be set to 1, and the rest should be set to zero. Fig. 6 depicts the possible selection vectors as well as the affected services by each change option and the values of three Boolean coefficients (which will be discussed later in this subsection).

Analyzing Costs of Change in SVM: We define the total cost of reflecting the changes of BPFM to SVM as a function of (1) the costs due to the changes in service interfaces ($ICost$) and (2) the costs imposed by the changes in relationships between the services ($RCost$).

Since the calculation of economical costs of change is not within the scope of this research, we consider the change cost as the number of affected artifacts. Here, artifacts are classified into two main classes: first, the domain-level concrete services, and second, the application-level products that use the domain-level services in their configurations.

Therefore, *ICost* imparts two costs: costs due to changes of (domain-level) concrete services developed internally or consumed from external repositories that implement and realize the abstract architecture-level services that are subject to change; and second, costs of change to all (application-level) configured products derived from the product line, which invoke those affected concrete services. The interface cost is calculated in formula (4) in which, z denotes a solution (a potential ΔS), and No_{csi} and No_{appi} represent the number of concrete services and the applications affected by the changes to the interface, respectively.

Similarly, *RCost* can be calculated. However, changes that are imposed by changing the relationships between services do not impose any effects on the domain-level concrete services; however, the products are affected, since it is crucial to keep the products "conforming" to the domain level, i.e., the new constraints (relationships) should hold in each product configuration. Otherwise, products will deviate from the product line, and this imposes extra costs for maintaining the product line. Hence, since product reconfiguration may be required, we will have formula 5, in which, z denotes a potential solution and No_{appr} represents the number of application that need to be reconfigured due to the changes to the relationships between services. Furthermore, changes to relationships in BPFM, require a re-extraction of relationships in SVM. This is achieved by Algorithm 1 , which uses the possible configurations that can be derived from the BPFM and the set of services (from the SVM) to generate the SVM which is modified in its variability relationships (we proposed this algorithm in a previous work [18], which aimed at automating extraction of an SVM from a BPFM).

$$ICost(z) = No_{csi} + No_{appi} \tag{4}$$

$$RCost(z) = No_{appi} \tag{5}$$

A change item, regarding its effect on SVM, may result in changes to either the concrete services, the products, both or neither (either impacting their interface or relationships or both). Therefore for each change item, we can define Boolean coefficients $c1$, $c2$ and $c3$, which show whether the change necessitates considering costs for changing interfaces of concrete services ($c1$), costs for changing products because of interface changes in concrete services ($c2$), and costs for reconfiguring the products due to the changes in the relationships ($c3$). Table 2 lists the values of $c1$ to $c3$ per each potential feasible change scenario in SVM. These coefficients for our running OLS example are listed for each change option in Fig. 6.

For a set of potential solutions, Z, we can form a set Obj_z, which is populated with individual objective vectors as ordered pairs of $(ICost, RCost)$ to be minimized. We will use these objective vectors to find a set of the most cost-

effective change options in the target SVM in response to the changes in the corresponding BPFM.

Algorithm 1 Service Variability Identification

Input: Service Set (SS), BPFM Configuration Set (CS)
Output: Service variability Model (SVM)

```
1: SVM ← NIL
2: for each service in SS do
3:    for each c in CS do
4:       if !(IsUsed(service,c)) then
5:          // service is not used in configuration c
6:          add service to SVM.OptionalServiceSet
7:          ServiceIsOptional ← True
8:          break
9:       end if
10:      if !ServiceIsOptional then
11:         add service to SVM.MandatoryServiceSet
12:      end if
13:   end for
14:   for each s_i in optionalServiceSet do
15:      for each s_j ≠ s_i in optionalServiceSet do
16:         for each c in CS do
17:            if !((IsUsed(s_i,c) ⊕ (IsUsed(s_j,c))) then
18:               AreAlt ← false
19:            end if
20:            if !((IsUsed(s_i,c) ∨ (IsUsed(s_j,c))) then
21:               AreOR ← false
22:            end if
23:            if !(!(IsUsed(s_i,c) ∨ (IsUsed(s_j,c))) then
24:               AreIncl ← false
25:            end if
26:         end for
27:         if !(AreAlt = false) ∧ (s_j, s_i) ∉ SVM.AltRelations then
28:            add (s_i, s_j) to SVM.AltRelations
29:         end if
30:         if !(AreOr = false) ∧ (s_j, s_i) ∉ SVM.OrRelations then
31:            add (s_i, s_j) to SVM.OrRelations
32:         end if
33:         if !(AreIncl = false) then
34:            add (s_i, s_j) to SVM.IncRelations
35:         end if
36:      end for
37:   end for
38: end for
39: return SVM
```

Determining the Most Cost-Effective solution: In this step, for each change item in ΔB we should determine which change options are the least

costly. For this purpose, we utilize multi-objective Pareto optimization [19]. If there is a solution z, whose $ICost$ and $Rcost$ are both less than all other change options' $ICost$ and $RCost$ respectively, then z is announced the most cost-effective solution which dominates all other solutions in cost-effectiveness. However, the problem arises when we reach a set of solutions, in which, some solutions have lower $ICost$s but higher $RCost$s, or have higher $ICost$s but lower $RCost$s compared with the other ones. In this case, no single solution can be announced as "dominating"; instead, we only can select a set of solutions, that although not dominating other solutions, are not dominated by other solutions either; that is, the set of "non-dominated" solutions that form the "Pareto-optimal" set of solutions [19]. The results of this step are used in the next step for decision-making on the Pareto-optimal set of solutions. Algorithm 2 represents the triple steps of change impact analysis.

For our OLS example, considering the concrete services and the configured products using them as denoted in Table 3, the objective vectors are those represented in Table 4. Since solution z_2 (see Fig. 6) is selected as the Pareto optimal solution, and assuming that the change is approved by the decision makers, the changes are executed and the new version of SVM (i.e., version $i+1$) is depicted in Fig. 7. In the new version of SVM, operation "$Oper4$" is removed from service "$Serv2$", operation "$Oper7$" in "$Serv4$" has changed from mandatory to optional, and a new operation "$Oper9$" is added to "$Serv2$" (there is no need to remind that each operation "$Oper\ i$" is an operation to realize activity i in the BPFM). No change was made to the variability attributes of the services, and no relationship exists among the services, and thus the new SVM cannot be inconsistent.

Table 2: General change options in SVM due to the change items in BPFM and the corresponding coefficient values

No.	Change Item in BPFM	Change Option in SVM	$c1$	$c2$	$c3$
1	Remove a mandatory activity	Remove mandatory operation from the mandatory service, and the service does not get empty to be removed	1	1	1
2		Remove mandatory operation from the mandatory service, and the service gets empty and is removed	0	1	1
3	Remove a non-mandatory (optional or variant) activity	Remove non-mandatory operation from the mandatory service	1	1	0
4		Remove non-mandatory operation from the optional service, and the service does not get empty to be removed	1	1	1
5		Remove non-mandatory operation from the optional service, and the service gets empty and is removed	0	1	1
6	Add a new mandatory activity	Add a new mandatory operation to a mandatory service	1	1	0
7		Add a new mandatory operation to an optional service	1	1	0
8		Add a new mandatory operation to a new empty service	1	1	0
9	Add a new non-mandatory (optional or variant) activity	Add a new optional operation to a mandatory service	1	0	0
10		Add a new optional operation to an optional service	1	0	1
11		Add a new optional operation to a new empty service	1	0	1
12	Modify data of an existing activity	Modify the operation in the same service that it currently resides	1	1	0
13	Remove a constraint (relationship) b/w activities	(has no impact on service interfaces and their relationships)	0	0	0
14	Add a new constraint (relationship) b/w activities	(has impact on relationships between services)	0	0	1
15	Modify cardinality from (x,y) to (x',y') where $x' \leq x$ and $y' \geq y$	(has no impact on service interfaces and their relationships)	0	0	0
16	Modify cardinality from (x,y) to (x',y') where $x' > x$ or $y' < y$	(has impact on relationships between services)	0	0	1
17	Modify the variability attribute of an activity from mandatory to optional	Modify variability attribute of the operation from mandatory to optional, if the service remains mandatory	0	0	0
18		Modify variability attribute of the operation from mandatory to optional, if the service becomes optional	0	0	0
19	Modify the variability attribute of an activity from optional to mandatory	Modify variability attribute of the operation from optional to mandatory, if the service turns from optional to mandatory	0	0	1
20		Modify variability attribute of the operation from optional to mandatory, if the service was mandatory before change	0	0	0
21	Move activity outside variant regions	(has no impact on service interfaces and their relationships)	0	0	0
22	Move activity from outside to inside variant regions or vice versa	(has impact on relationships between services)	0	0	1

Algorithm 2 Change Impact Analysis

Input: $BPFM$, $BPFM'$, SVM
Output: Pareto optimal set of change solutions($POSC$)

1: $POSC \leftarrow NIL$
2: $\Delta B \leftarrow CalculateDelta(BPFM, BPFM')$
3: $Obj \leftarrow NIL$
4: **for** each CI_i in ΔB **do**
5: $COi \leftarrow$ all potential change options to reflect CI_i in SVM ()
6: fetch coefficients()
7: **end for**
8: Generate Z //all possible solutions
9: **for** each z_i in Z **do**
10: $ICost \leftarrow$ Calculate_ICost(z_i)
11: $RCost \leftarrow$ Calculate_RCost(z_i)
12: $Obj \leftarrow Obj \cup (Icost, RCost)$
13: **end for**
14: $POSC \leftarrow$ FindPaertoOptimalSet(Z , Obj)
15: **return** $POSC$

Table 3: Concrete services and currently existing products (before change)

abstract service in SVM	**concrete services**	**existing configurations (products)**
$S1$	$CS1$	$app1, app2$
$S2$	$CS2$	$app1, app2, app3, app4$
$S3$	$CS3$	$app3$
$S4$	$CS4$	$app1, app2$
	$CS5$	$app3, app4$

Table 4: Values of Objective functions for the potential solutions

Obj	**ICost**	**RCost**	**Selected in Pareto Optimal Set**
$Obj(z1)$	6	4	-
$Obj(z2)$	5	4	✓
$Obj(z3)$	6	4	-
$Obj(z4)$	6	4	-
$Obj(z5)$	6	4	-

3.3 Change Planning and Decision Making

In this step, domain architects and engineers decide between change options among the Pareto-optimal set of change options, and may even disapprove a change item to take place in the product line.

As stated earlier, change cost is not necessarily the main and single factor to consider when deciding on a change; on the contrary, there can be a multitude

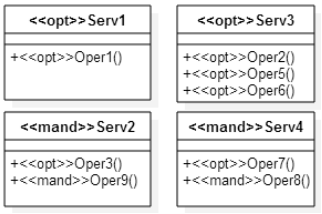

Fig. 7: New version of SVM (version $i + 1$)

of other factors such as market state, specific metrics and measures, and other particular features that exists especially for the external services; or other costs and benefits, and miscellaneous priorities that affect the decision, which cannot necessarily be grabbed in a formula to be calculated and included in the optimization process. That is why we emphasize on the human factor and the role the experts play in this regard. Therefore, a fully-automated method is not advisable for propagating the changes from the business process variability model to the abstract and concrete services and to the applications configured based on them; since it is unlikely to identify all effective factors. Details of this step is outside the scope of this research and can be achieved by utilizing different decision making processes such as AHP and its variants.

3.4 Change Execution

A potential way to execute the changes that are determined in the selected change items (ΔS) is using model driven methods, and more specifically, model transformation. The advantage of applying these methods is preserving consistency in the target models by setting consistency constraints on the meta-model (e.g. in OCL) that will prevent inconsistent models to form. Using the change items in ΔS, the modified version of SVM, SVM$'$, is formed.

3.5 Version Control

For supporting traceability between the versions of the product line, we utilize an evolution graph (similar to what is proposed in [20]) as follows: in this graph, the nodes denote different product line versions, and directed edges denote a transition from a version to another. The nodes are attributed with the version number, and other version-related attributes such as creation date, approver, etc., while the edges are attributed with ΔB, ΔS, rationale for transitioning from the previous version to the new one, decisions made, etc. Every approved change in the product line domain must be reflected to and documented in the evolution graph, which acts as a shared knowledge among the four operations. Apache SVN can be utilized as a well-known version (revision) control tool.

4 Evaluation

To evaluate the method, we utilized the GQM approach [21] and controlled experimentation, performed on three experimental cases. In GQM, evaluation goal, evaluation questions and metrics that are measured to answer the evaluation questions are determined. We define the evaluation goal, questions and metric as follows:

Evaluation Goal (G): To analyze the proposed method for the purpose of evaluation from the aspect of specific quality metrics of cost minimization and consistency preservation in linear evolution in service-oriented product lines.

Evaluation Questions (Q): *Q1*: Does the method minimize change costs?; *Q2*:Does the method preserve consistency of the SVM after applying the change items, in terms of avoiding logical conflicts?

Evaluation Metrics (M): To answer question *Q1*, we calculate change ratio which is the ratio of artifacts changed due to interface change and due to relationship change of the services (Metrics *M1* and *M2*). Moreover, to answer question *Q2*, we introduce consistency degree metric (*M3*) which calculates a ratio of the number of consistent changed SVMs to the total number of changed SVMs that had gone through the change process. Table 5 represents more GQM details of evaluation.

Table 5: GQM details of evaluation

Evaluation Goal	Evaluation Question	Evaluation Metric	Metric Name	Metric Formula	Description
$G1$	$Q1$	$M1$	Change Ratio for Interface Change (CRI)	$CRI = \frac{AAI}{A}$	AAI: number of affected artifacts by interface change A: total number of artifacts
		$M2$	Change Ratio for Relationship Change (CRR)	$CRR = \frac{AAR}{A}$	AAR: number of affected artifacts by relationship change A: total number of artifacts
	$Q2$	$M3$	Consistency Degree (CD)	$CD = \frac{CS}{S}$	CS: number of consistent changed SVMs S: total number of changed SVMs

4.1 Experimentation materials and procedure

Ten product lines (A to J) were selected for experimentation, for each of which a distinct BPFM was derived beforehand based on business processes existing in detergent manufacturing, sales and distribution, and food industries, by domain engineers and business experts. Input materials to the experimentation were the current versions of SVMs related to each case, along with their corresponding BPFM models. Other information such as the number of concrete services and the number of applications was also supplied.

Then, to design changes to the BPFM models, a set of random numbers were generated, to designate the number of changes to that BPFM (which we considered between 1 to 8) and the change scenarios for each change (random numbers between 1 and 22). If a change scenario was not applicable on the BPFM, then (rotationally) the next possible scenario was chosen. Moreover, we sometimes re-generated the random numbers for scenarios to make sure we had covered each change scenario at least once. All random numbers were generated in MS Excel.

Then, the changed BPFMs were designed in a way that the change requirements denoted by the numbers and scenarios of changes (random numbers) were met. Some details about the experimental materials are listed in Table 6.

The metrics were measured on the new SVMs generated by the method. Moreover, the consistency of the obtained outputs was then inspected using the SPLOT online consistency checking tool for variability models [22].

Table 6: Details on experimental cases for evaluating the method

Exp.Case	Business process	Business domain	No. of changes in BPFM	Change scenarios	No. of total concrete services	Total No. of Configurations
A	Purchase order processing	Procurement	5	10,18,5,17,14	25	138
B	Order Processing	Sales	3	2,19,13	10	45
C	Adding new asset	Asset management	7	6,1,20,17,10,14,3	9	45
D	Repair order processing	Maintenance, operation and repair	2	19,21	18	107
E	Online sales	Sales	6	11,17,6,21,1,14	7	45
F	Letter submission	Office automation	2	16,20	4	29
G	Issue warehouse note	Warehousing	4	11,9,22,4	4	22
H	Create new bill of materials	Manufacturing	2	7,12	9	49
I	Calculate wages and salaries	Payment	1	11	14	94
J	Issue accounting notes	Accounting	6	5,9,21,8,3,15	6	39

4.2 Experimentation Results

The summary of experimentation results to answer question *Q1* is shown in Table 7. As it is obvious, average change ratios of 33.6% and 33% are resulted by the method for changes costs caused by interface changes and by relationship changes, respectively. Therefore, in response to question *Q1*, the method has managed to affect an average of only 33.3% (one third) of the artifacts in the product line.

Since the Pareto optimal set, which was obtained by the method is a set of optimal non-dominated solutions, it is obvious that these solutions are the least costly from the aspects of interface and relationships change costs, compared with the other solutions that are not a member of this set. Therefore, if the decision maker selects any of the members of the Pareto optimal set, he or she has chosen one of the least costly choices.

To answer question *Q2*, we measured metric *M3*, namely, the consistency degree, which shows a full (100%) consistency preservation. The reason is that the variability relationships between services must be re-determined by Algorithm 1 when a change occurs to the relationships; and the algorithm guarantees consistency. In other words, the automated process prevents formation of conflicting variability relationships between services, unless a human factor intervenes in the produced results. Hence, the answer to question *Q2* would be that, in case there is no human intervention in the change results, the method guarantees 100% consistency preservation.

Table 7: A summary of experimentation results for Question Q1

Experimental Case	No. of Total Artifacts	$ICost$	$RCost$	CRI (%)	CRR (%)
A	163	27	26	17	16
B	55	17	24	31	44
C	54	24	42	44	78
D	125	0	28	0	22
E	52	6	32	12	62
F	33	0	4	0	12
G	26	21	22	81	85
H	58	30	0	52	0
I	108	1	0	1	0
J	45	44	5	98	11
Average	71.9	17	18.3	33.6	33

5 Conclusion

In this paper, focusing on linear evolution scenario in service-oriented product lines, we dealt with the problem of propagating changes from domain requirements (the business process variability model) to the domain architecture (the service variability model) so that the costs of change in the concrete domain services and the configured applications are minimal; and simultaneously, the consistency of the target domain architecture model is preserved. This paper provides a method based on Pareto optimization to find an optimal solution, that is, the least costly changes in the target architecture model. The method consists of four main operations, namely, change identification, change impact analysis, change planning and change execution, interrelating with the evolution graph, which is used for version control. These operations along with the evolution graph form the whole evolution cycle to control and manage linear evolution. The evaluation results showed that the method manages to minimize change costs and can fully preserve consistency of the target model.

The future research includes proposing methods to cover other evolution scenarios, implementing model transformation to realize change execution as well as model-driven method to apply the deltas, along with extending the method to support runtime changes and dynamic reconfiguration of service-based applications in the service-oriented product line.

References

1. Clements, P., Northrop, L.: Software product lines: practices and patterns. Addison-Wesley Reading (2002)
2. Gomaa, H.: Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures. Boston: Addison-Wesley (2004)
3. Van der Linden, F., Pohl, K.: Software Product Line Engineering: Foundations, Principles, and Techniques. Stuttgart: Springer (2005)
4. Erl, T.: Service-oriented architecture (SOA): concepts, technology, and design. Prentice Hall Englewood Cliffs (2005)

5. Wu, Y., Peng, X., Zhao, W.: Architecture Evolution in Software Product Line: An Industrial Case Study. In: Schmid, K. (ed.) Top Productivity through Software Reuse, vol. 6727, pp. 135-150. Springer Berlin Heidelberg (2011)
6. Kephart, J., Kephart, J., Chess, D., Boutilier, C., Das, R., Kephart, J.O., Walsh, W.E.: An architectural blueprint for autonomic computing. IEEE internet computing 18, (2007)
7. Moon, M., Hong, M., Yeom, K.: Two-level variability analysis for business process with reusability and extensibility. In: 32nd Annual IEEE International Conference on Computer Software and Applications, COMPSAC'08, pp. 263-270. IEEE, (2008)
8. Park, J., Moon, M., Yeom, K.: Variability modeling to develop flexible service-oriented applications. Journal of Systems Science and Systems Engineering 20, 193-216 (2011)
9. Laguna, M.A., Crespo, Y.: A systematic mapping study on software product line evolution: From legacy system reengineering to product line refactoring. Science of Computer Programming 78, 1010-1034 (2013)
10. Peng, X., Yu, Y., Zhao, W.: Analyzing evolution of variability in a software product line: From contexts and requirements to features. Information and Software Technology 53, 707-721 (2011)
11. Ajila, S.A., Kaba, A.B.: Evolution support mechanisms for software product line process. Journal of Systems and Software 81, 1784-1801 (2008)
12. Hinchey, M., Park, S., Schmid, K.: Building dynamic software product lines. Computer 22-26 (2012)
13. Acher, M., Cleve, A., Collet, P., Merle, P., Duchien, L., Lahire, P.: Extraction and evolution of architectural variability models in plugin-based systems. Software & Systems Modeling 1-28 (2013)
14. Andrikopoulos, V., Bucchiarone, A., Di Nitto, E., Kazhamiakin, R., Lane, S., Mazza, V., Richardson, I.: Service Engineering. In: Papazoglou, M., Pohl, K., Parkin, M., Metzger, A. (eds.) Service Research Challenges and Solutions for the Future Internet, vol. 6500, pp. 271-337. Springer Berlin Heidelberg (2010)
15. Alférez, G.H., Pelechano, V., Mazo, R., Salinesi, C., Diaz, D.: Dynamic adaptation of service compositions with variability models. Journal of Systems and Software 91, 24-47 (2014)
16. Pleuss, A., Botterweck, G., Dhungana, D., Polzer, A., Kowalewski, S.: Model-driven support for product line evolution on feature level. Journal of Systems and Software 85, 2261-2274 (2012)
17. Gomaa, H., Hashimoto, K.: Dynamic software adaptation for service-oriented product lines. In: Proceedings of the 15th International Software Product Line Conference, Volume 2, pp. 35. ACM, (2011)
18. Khoshnevis, S.: Identifying Variable Services for Multi-Tenant SaaS Applications as a Service-Oriented Product Line Using MOEA/D. International Journal of Software Engineering and Its Applications (2015)
19. Coello, C.C., Lamont, G.B., Van Veldhuizen, D.A.: Evolutionary algorithms for solving multi-objective problems. Springer (2007)
20. Pressman, R., S.: Software engineering: a practitioner's approach. McGrow-Hill International Edition (2005)
21. Van Solingen, R., Basili, V., Caldiera, G., Rombach, H.D.: Goal question metric (GQM) approach. Encyclopedia of Software Engineering (2002)
22. Mendonca, M., Branco, M., Cowan, D.: SPLOT: software product lines online tools. In: Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications, pp. 761-762. ACM, (2009)