Modeling and Managing Variability in Process-Based Service Compositions

Tuan Nguyen, Alan Colman, and Jun Han

Faculty of Information and Communication Technology,
Swinburne University of Technology, Melbourne, Australia
{tmnguyen,acolman,jhan}@swin.edu.au

Abstract. Variability in process-based service compositions needs to be explicitly modeled and managed in order to facilitate service/process customization and increase reuse in service/process development. While related work has been able to capture variability and variability dependencies within a composition, these approaches fail to capture variability dependencies between the composition and partner services. Consequently, these approaches cannot address the situation when a composite service is orchestrated from partner services some of which are customizable. In this paper, we propose a feature-based approach that is able to effectively model variability within and across compositions. The approach is supported by a process development methodology that enables the systematic reuse and management of variability. We develop a prototype system supporting extended BPMN 2.0 to demonstrate the feasibility of our approach.

Keywords: Process variability, service variability, variability management, service composition, feature modeling, model mapping, Software Product Line (SPL), Model Driven Engineering (MDE).

1 Introduction

Process-based service compositions are efficient approaches for developing composite services and applications using process modeling techniques. The two de facto standards for this purpose are BPMN (Business Process Modeling Notation) for modeling purposes and BPEL (Business Process Execution Language) for execution purposes. Generally, in both techniques, each composite service is described by a process model which specifies the flow of activities (i.e. *control flow*), the interaction between the process and partner services (i.e. *message flow*), and the way data is moved throughout the process (i.e. *data flow*).

Due to the diversification and the personalization of service consumption, service variability has become an important factor in the lifecycle of service development [1, 2]. Service variability is defined as the ability of a service/process to be efficiently extended, changed, customized or configured for use in a particular context [3]. Such variability can originate from a service provider wishing to provide different versions of the same service for different market segments or with different pricing models, or from service consumers wishing to customize a service to match their particular business requirements.

Service variability brings about a new type of service, namely *customizable service,* in service ecosystems [4]. *A customizable service* is a service whose *runtime customization* by a consumer will result in a particular service variant matching the consumer's requirements [5-7]. For services with a large number of service variants, the deployment of customizable services, instead of conventional services, will much benefit service consumers. This is because there is disadvantage with either deploying an all-in-one non-customizable service or deploying all service variants separately. In the first case, the resulting non-customizable service has a large service description most of which is not relevant to one particular consumer. In the second case, it is difficult for service consumers to recognize the similarity and difference among those service variants in order to select the most appropriate one [2].

Modeling and managing variability in customizable composite services are challenging. There are two key concerns that need to be addressed [8]. Firstly, how to model *variation points* and *variants*? Secondly, how to capture *dependencies* among variabilities? Variability dependencies describe such relationships as the binding of variants at one or several variation points requires or excludes the binding of variants at other variation point(s). We identified in our previous work that, in the service computing context, besides *variability intra-dependencies* which represent dependencies within a service composition, there are *variability inter-dependencies* which represent dependencies between the composition and its customizable partner services [9]. Variability inter-dependencies reflect the situation when the runtime resolution of variability in the composition requires the runtime resolution of variability at partner services. And this process may also cause a *ripple effect* in the service ecosystem since service composition is recursive.

In terms of variability management, Software Product Line (SPL) is a successful paradigm that builds upon techniques for systematic identification and management of variability [10]. Many related efforts have exploited concepts and techniques from SPL in addressing variability in process-based service compositions, e.g. [11-14]. These approaches are able to capture variability and variability dependencies within the control flow and the data flow of a process model. However, all these efforts fail to capture *variability inter-dependencies*. Consequently, these approaches are not capable of managing variability in such service compositions that are aggregated from customizable partner services.

To address this problem, we propose a comprehensive approach to modeling and managing variability in process-based service compositions. In particular, we extend the BPMN 2.0 metamodel to incorporate variation points and variants with respect to not only control flow and data flow but also message flow. We then extend a feature modeling technique from SPL to capture variability dependencies within and across service compositions. We also specify a process development methodology that elaborates how to systematically model and manage variability at design time, as well as instantiating variability at runtime. The methodology builds upon Model Driven Engineering (MDE) techniques to automate large parts of its operations.

The structure of the paper is as follows. Section 2 presents a discussion of related work. In section 3, we describe a motivating scenario, followed by the explanation of techniques underpinning our research in section 4. Section 5 presents our approach to modeling variability and variability dependencies. We describe an approach to developing service compositions with managed variability in section 6. The prototype system is described in section 7 before our conclusion of the paper in section 8.

2 Related Work

A number of works has been proposed for modeling and managing variability in process-based service compositions [1, 2, 11-16]. In general, they can be classified into two categories.

The first category consists of work that aims to extend BPEL [12, 13, 15]. In particular, Chang [15] and VxBPEL [12] extend the XML schema for BPEL in order to incorporate information about variation points and variants into the business process definition. In contrast, Mietzner [13] uses a separate variability descriptor to define the location of variation points in the business process definition and possible variants. In general, the advantage of extending BPEL is that an executable process variant can be automatically derived by resolving all variation points. However, VxBPEL and Chang's work suffer from tangled and scattered business process definitions. Mietzner's work overcomes this problem by using a separate variability description. Nevertheless, since variability is modeled at the implementation level, these approaches become very complex due to the large number of variation points.

Work in the second category focuses on extending process models described using BPMN or UML Activity diagrams [1, 2, 11, 14, 16]. The general approach for these efforts is to extend the process metamodel so that variation points and variants can be explicitly introduced. Since variability is modeled at the architectural level, the number of variation points is much smaller than the ones at the process definition level. Therefore, these approaches overcome the complexity issue of the ones in the first category. However, except [14, 16], all other works only focus on variation points and variants with respect to the control flow of process models. Works in [14, 16] takes a step further to consider the data flow as well. Consequently, only these works can support the derivation of executable process variants.

Although *variability intra-dependencies* have been considered in most of the related work, e.g. [2, 12], the major issue with work in both categories is that they are not able to capture *variability inter-dependencies*. All work builds on an assumption that all partner services are not customizable. Consequently, those approaches are not applicable to composite services orchestrated from partner services some of which are customizable.

3 Motivating Scenario

A Content Management System (CMS) Provider wants to develop a composite service which allows various Content Providers to post news entries (cf. Figure 1).

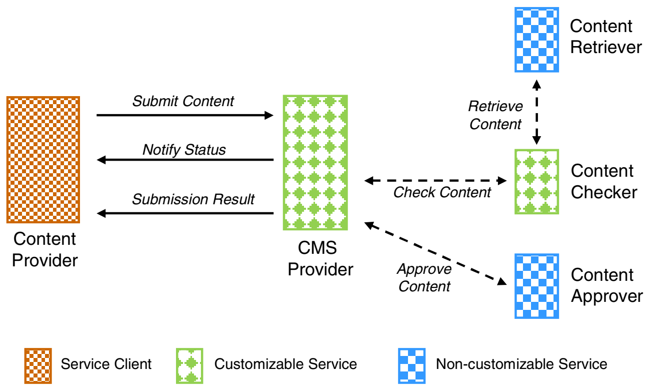


Fig. 1. A news posting composite service

Due to different requirements from Content Providers, the CMS Provider will support the following variability in its business process:

- Content Providers may choose to directly send news entries or specify an external URL resource as the content source.
- Content Providers may also opt to receive posting status update from the CMS Provider.

There are many services available that the CMS Provider may reuse in implementing its process. For example, there are services for checking the correctness of the news entry (e.g. grammar check) and there are services for approving the news submission (e.g. checking the publishing policies). In this case study, the CMS Provider will utilize two of those services, namely ContentChecker service and ContentApprover service. While the ContentApprover service is a *non-customizable service* accepting the news content and returning the approving result, the ContentChecker service is a *customizable service*. It has two service variants. The first service variant accepts the news content, performs the checking and then returns the result. The second variant accepts a URL and invokes the ContentRetriever service for retrieving the content before performing the checking. The utilization of this ContentChecker service frees the CMS Provider from the overhead of retrieving the content in a case a URL is provided from a Content Provider. Consequently, variability in the CMS Provider will depend on the variability in the ContentChecker.

4 Underpinnings of Our Approach

In this section, we explain the techniques that underpin our approach. In particular, we briefly describe feature modeling techniques from SPL, our solution for describing variability of customizable services based on the concept of features, and how the service variability description is utilized to support runtime service customization.

4.1 Feature Modeling Technique

Feature modeling are techniques in SPL for capturing the commonalities and differences among a family of software products [17]. Features are visible characteristics that are used to differentiate one family member from others. A feature model is represented as a hierarchically arranged set of features with *composed-by* relationship between a parent feature and its child features. In addition, there are cross-tree *constraints* that typically describe inclusion or mutual exclusion relationships. A feature model is an efficient abstraction of variability and provides an effective means for communicating variability between different stakeholders. In addition, it helps to drive the design and the development of variability throughout all stages of the product line development.

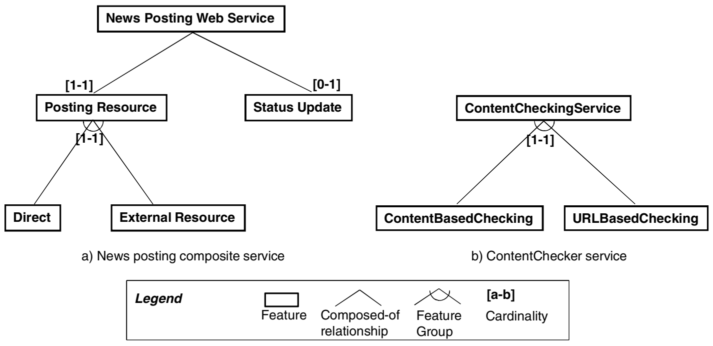


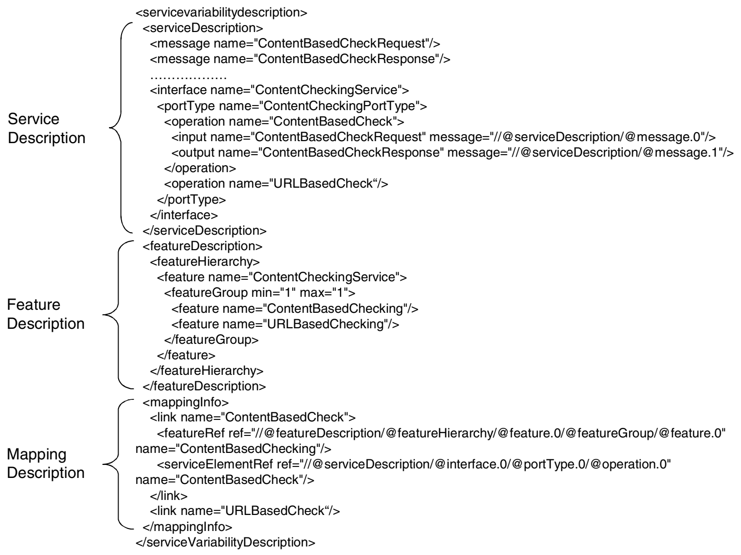
Fig. 2. Examples of feature model

While there are many manifestations of feature modeling techniques, e.g. [18-20], in our work we exploit Czarnecki's cardinality-based feature modeling technique [21]. The main reason for this choice is that the concepts of *feature cardinality* and *group cardinality* well suit the needs of service customization. A *feature cardinality*, associated with a feature, determines the lower bound and the upper bound of the number of the feature that can be part of a product. A *group cardinality*, associated with a parent feature of a group of features, limits the number of child features that can be part of a product when the parent feature is selected.

Figure 2a demonstrates a feature model representing variability of the news posting composite service. Based on their cardinality, *"Posting Resource"* is a mandatory feature, while *"Status Update"* is an optional feature. In addition, *"Posting Resource"* is a group of alternative features. It means that, all consumers need *"Posting Resource"* capability, which can be either *"Direct"* or *"External Resource"*, while they can opt to have *"Status Update"* capability when consuming the service. Similarly, Figure 2b demonstrates the feature model for the ContentChecker service. Its variability is represented as a group of two alternative features, *"ContentBasedChecking"* and *"URLBasedChecking"*.

4.2 Feature-Based Service Variability Description

In order to describe variability of a services and facilitate service customization, we define a new language, namely WSVL (Web Service Variability description Language), based on the concept of features. Due to space limitation, we briefly describe the language through the example of the ContentChecker service without going into the detail of motivations and requirements behind it. The language (cf. Figure 3) has three parts. Firstly, the *ServiceDescription* part describes the capability of the service and represents the superset of the capability of all service variants. In this scenario, the service description consists of two operations, *ContentBasedCheck* and *URLBasedCheck*, using different message formats for realizing two service variants. The service description is only expressed at the abstract level for modeling purposes. Once a service variant is derived, its complete service description, described in WSDL, will be generated. Secondly, the *FeatureDescription* part describes the variability of the service in term of features. It is actually the serialization of the feature model for the corresponding service (cf. Figure 2b). And thirdly, the *MappingDescription* part describes the mapping from variant features in the feature description part to variable capability in the service description part as a set of links. For example, the first link shows the mapping between the feature *"ContentBasedChecking"* and the corresponding operation, *"ContentBasedCheck"*. In general, a link represents 1-to-m mapping between a feature and service capabilities. The service variability description provides information on what capability of the corresponding service is available in a service variant given a feature configuration[1].


Service Description
Feature Description
Mapping Description

```
<servicevariabilitydescription>
 <serviceDescription>
  <message name="ContentBasedCheckRequest"/>
  <message name="ContentBasedCheckResponse"/>
  ..................
  <interface name="ContentCheckingService">
   <portType name="ContentCheckingPortType">
    <operation name="ContentBasedCheck">
     <input name="ContentBasedCheckRequest" message="//@serviceDescription/@message.0"/>
     <output name="ContentBasedCheckResponse" message="//@serviceDescription/@message.1"/>
    </operation>
    <operation name="URLBasedCheck"/>
   </portType>
  </interface>
 </serviceDescription>
 <featureDescription>
  <featureHierarchy>
   <feature name="ContentCheckingService">
    <featureGroup min="1" max="1">
     <feature name="ContentBasedChecking"/>
     <feature name="URLBasedChecking"/>
    </featureGroup>
   </feature>
  </featureHierarchy>
 </featureDescription>
 <mappingInfo>
  <link name="ContentBasedCheck">
   <featureRef ref="//@featureDescription/@featureHierarchy/@feature.0/@featureGroup/@feature.0"
name="ContentBasedChecking"/>
   <serviceElementRef ref="//@serviceDescription/@interface.0/@portType.0/@operation.0"
name="ContentBasedCheck"/>
  </link>
  <link name="URLBasedCheck"/>
 </mappingInfo>
</serviceVariabilityDescription>
```

Fig. 3. Service variability description for the ContentChecker service

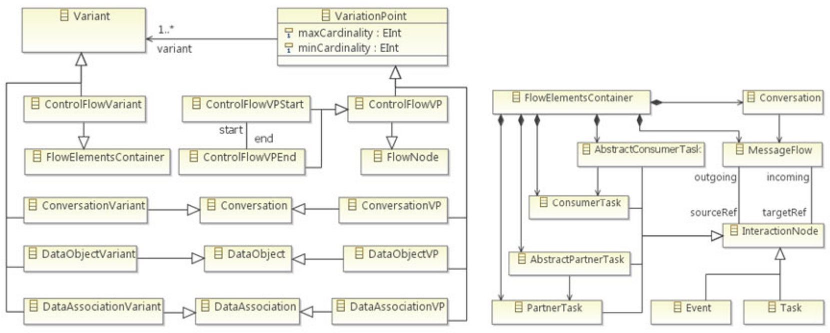
[1] A feature configuration is a specialized form of a feature model in which all variability is resolved, i.e. all variant features are selected or removed.

4.3 Feature-Based Service Customization Framework

In previous work, we developed a feature-based service customization framework that allows service consumers to customize a service at the business level [5]. In particular, based on the service variability description, service consumers can select features they need and unselect features they do not need. Feature selection has to conform to feature cardinality, group cardinality and constraints described in the feature model to generate a valid feature configuration. The feature configuration is then communicated back to the service provider so that the service provider can generate a service interface description and a service implementation bound to the service interface description. This service variant is then dynamically deployed to an endpoint so that the service consumer can invoke. In previous work, we have focused on how to model, manage and instantiate variability at the service interface level. The work in this paper complements that work in addressing the issues of how to model and manage variability in the service implementation (i.e. business process), and then generating a variant based on a particular feature configuration. In addition, the work in this paper also exploits that technique for customizing partner services.

5 Modeling Variability in Process-Based Service Compositions

As explained, variation points and variants need to be explicitly introduced into process models. To this end, there are two requirements for our approach. Firstly, the complexity in modeling variability needs to be alleviated. Modeling variability in business processes is challenging because of the existence of a large number of variation points and variants. Therefore, it is important to reduce the number of variation points and variants that need to be considered. Secondly, the approach needs to support variability instantiation, i.e. the runtime derivation of executable process variants for customization purposes.



a) Variation points and variants b) PartnerTask and ConsumerTask

Fig. 4. Process metamodel extension

In order to satisfy these two requirements, we have decided to model and manage variability within process models described by BPMN. The advantage of using BPMN or UML Activity diagram over BPEL is that the number of variation points and variants within a BPEL definition is much more than the ones within a BPMN or UML model. And we selected BPMN over UML Activity because of its wide acceptance and well support. At the time we developed our approach, BPMN 2.0 has been released and it provided a sufficient metamodel for modeling process-based service compositions. However, there is no significant difference between the two. One can easily apply the solution we present here over to UML Activity diagrams and achieve similar results. In addition, we exploit MDE techniques in our approach to automate large parts of the solution and facilitate not only variability management but also variability instantiation.

5.1 Extending BPMN for Representing Variation Points and Variants

Our key idea for introducing variability modeling capability into process modeling is to define a general metamodel for variation points and variants, then weave this metamodel into the BPMN 2.0 process metamodel to make it capable of supporting variability. The extension will focus on all three aspects of service compositions: control flow, data flow, and message flow. In addition to modeling variability in the control flow and data flow as done in related work, our approach takes a further step to capture variability in the message flow as well. Hence, the approach is capable of not only supporting executable process variant derivation, but also capturing variability inter-dependencies. The result of this is shown in Figure 4.

The general variability metamodel is composed of two elements: *VariationPoint* and *Variant*. A *VariationPoint* represents any place in the process model where variability can occur. Each *VariationPoint* is associated with a set of *Variants* from which one or several will be bound to *VariationPoint* when the variability is resolved. The attributes *minCardinality* and *maxCardinality* define how many *Variants* should be bound to one *VariationPoint*. These attributes have the same semantics as the cardinality concept adopted in the feature modeling technique.

Variation point in the control flow can be interpreted as any location in the process model at which different execution paths can take place. Therefore, we introduce new *FlowNode* elements, namely *ControlFlowVP,* and its two direct inheritances, namely *ControlFlowVPStart* and *ControlFlowVPEnd*, for representing starting point and end point of each variability. Variants in the control flow can be arbitrary process fragments. Therefore, *ControlFlowVariant* is inherited from *FlowElementContainer*.

Variability in data flow can be considered as different ways for storing data (i.e. *DataObject*) or different ways for moving data around (i.e. *DataAssociation*). Since variants in data flow are usually alternative variants, we model both variation points and variants as inherited elements from the same element type. That is, for variability of *DataObject*, we define both variation points, i.e. *DataObjectVP*, and variants, i.e. *DataObjectVariant*, as inherited elements from *DataObject*. A similar approach applies with *DataAssociation*, *DataAssociationVP*, and *DataAssociationVariant*.

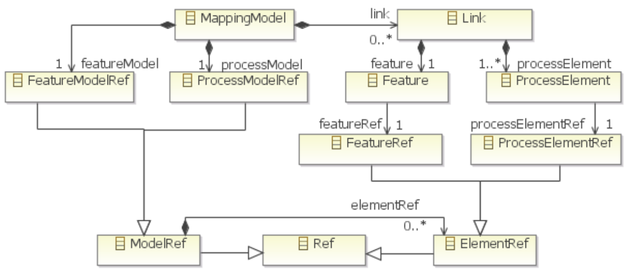


Fig. 5. Mapping metamodel

Variability in message flow can be seen as alternative *Conversations* between two parties, i.e. the process and a partner service (or a consumer). Therefore, in a similar fashion to modeling variability in data flow, we model both variation points, i.e. *ConversationVP*, and variants, i.e. *ConversationVariant*, as inherited elements from *Conversation*. In addition, we introduce new elements, namely *PartnerTask* and *AbstractPartnerTask* (cf. Figure 4b). A *PartnerTask* models a task performed by a partner service. An *AbstractPartnerTask* models a set of alternative *PartnerTasks* and it represents a variable capability provided by a partner service. The introduction of *PartnerTask* and *AbstractPartnerTask* facilitates the modeling of variability inter-dependencies since variability inter-dependencies will be the mapping between these elements and variant features of partner services, namely *variant partner features*. In a similar fashion, we introduce *ConsumerTask* and *AbstractConsumerTask* for the interaction between the business process and its consumers. These extended elements facilitate the generation of the service variability description for this service composition.

5.2 Modeling Variability Intra-dependencies

Variability intra-dependencies represent dependencies among variation points and variants within a process model. Therefore, the intuitive way for modeling variability intra-dependencies is to model variability constraints as elements of the process models. However, the disadvantage of this approach is twofold. Firstly, the resulting process model will be swamped with dependencies information and become too complex. Secondly, since variability intra-dependencies are embedded in process model definitions, it becomes harder to identify conflicts in such dependencies [22].

In fact, variability in the process model is the *realization of variability* in the feature model of the service composition. In other words, the identification and modeling of variation points and variant in a process are driven by variant features in the feature model. *Variability intra-dependencies among different variants come from the fact that those variants realize variant features.* Therefore, we exploit the model mapping technique for relating variation points and variants in the process model with variant features in the feature model. We refer to this type of mapping model as *FeatureTask mapping model*. Due to mentioned *realization relationships*, those mappings along with feature constraints in the feature model account for all variability intra-dependencies in the process model. In addition, this approach has the following advantages in comparison with embedding constraints in the process definition. On the one hand, it helps to separate variability constraint information from the process model, thus simplifies the definition. On the other hand, the validation of process configuration is led to the validation of a feature configuration, which is well-studied in SPL [23].

The mapping metamodel for this purpose is shown in Figure 5. A *MappingModel* relates variant features in a feature model, referenced by *FeatureModelRef*, with variants in a process model, referenced by *ProcessModelRef*. It is composed of *Links* and each *Link* consists of a *Feature* and at least one *ProcessElement*. *Feature* and *ProcessElement* reference elements in the feature model and the process model respectively. In this way, each *Link* enables a feature to be mapped to one or several variant process elements in the process model.

5.3 Modeling Variability Inter-dependencies

Variability inter-dependencies represent dependencies between variability in the process model and variability in partner services. Since variability of partner services can be described using feature models (cf. Figure 3), in a similar fashion as modeling variability intra-dependencies, we exploit the model mapping technique to model these dependencies. The main difference between the mapping model for variability intra-dependencies and the mapping model for variability inter-dependencies is the origin of variant features. While variability intra-dependencies is modeled with respect to variant features in the feature model of the service composition, variability inter-dependencies is modeled with respect to variant partner features.

In particular, a mapping model for variability inter-dependencies captures the correspondence between *PartnerTasks* within the process model and variant partner features. We refer to this mapping model as *PartnerTaskFeature mapping model*. It should be noted that between *PartnerTasks* and variant partner features, there does not exist a "natural" *realization relationship* as the ones for variability intra-dependencies. If the identification and modeling of *PartnerTasks* and *AbstractPartnerTasks* are driven by the variant partner features, such *realization relationships* establish. Otherwise, the identification and modeling of *PartnerTasks* and *AbstractPartnerTasks* are independent of variant partner features, and *realization relationships* may not exist. In this way, all variability inter-dependencies exist by chance. Therefore, high inter-dependency between the service composition and partner services represents high chance of reuse of service variability from partner services toward the service composition. In later section, we describe a process development methodology that systematically increases the chance of reuse of service variability.

Since variability in the feature model of the business process is mapped to variability in the process model, i.e. *FeatureTask mapping model*, and a part of variability in the process model, i.e. *PartnerTasks*, is mapped to variability in partner feature models, i.e. *PartnerTaskFeature mapping model*, it is possible to generate the mapping from variant features in the feature model of the service composition to variant partner features. This mapping model conforms to a similar mapping metamodel as Figure 5 and allows us to capture variability inter-dependencies at the highest level of abstraction, i.e. the feature level. We refer to this type of mapping model as *FeatureFeature mapping model*. In summary, there are two types of feature mapping models for representing variability inter-dependencies: *PartnerTaskFeature* and *FeatureFeature mapping models*.

6 A Bottom-Up Process Development Methodology

In this section, we describe a methodology for developing service compositions with systematic management and reuse of variability (cf. Figure 6). One key feature of the methodology is that, it increases the chance of reusing service variability provided by partner services. To this end, variability information from partner services is explicitly utilized in driving the identification and modeling of variability within the business process.

6.1 Overview

In the first activity, the capability of the service composition is modeled using the feature modeling technique. The result of this activity is a feature model capturing commonalities and variabilities of the composite service to be. Given a model of desired features, the next activity will be the selection of partner services that can be used for the service composition. There are two types of services that will be selected: *(conventional) non-customizable partner services* and *customizable partner services*. The explicit selection of customizable partner services helps to reduce overhead of addressing variability within the service composition. Customizable partner services come with service variability descriptions.

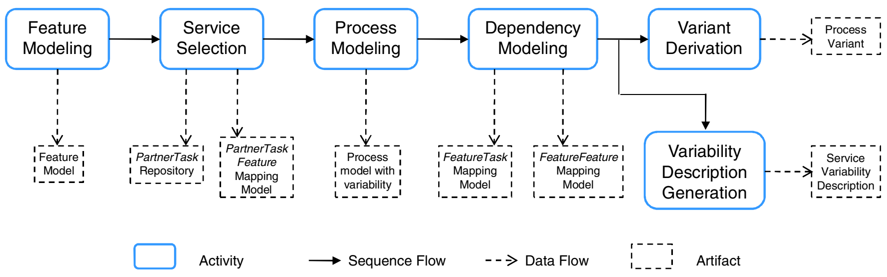

Feature Modeling
Service Selection
Process Modeling
Dependency Modeling
Variant Derivation
Process Variant
Feature Model
PartnerTask Repository
PartnerTask Feature Mapping Model
Process model with variability
FeatureTask Mapping Model
FeatureFeature Mapping Model
Variability Description Generation
Service Variability Description
Activity
Sequence Flow
Data Flow
Artifact

Fig. 6. A methodology for developing process-based service compositions

During the second activity, both non-customizable partner services and customizable partner services are transformed into a set of partner tasks that will be selectable for modeling the process. *A partner task is an operation provided by a partner service that is responsible for an atomic message flow between the partner* *service and the service composition.* While non-customizable partner services are transformed to a set of non-customizable partner tasks, results of transforming customizable partner services are sets of alternative partner tasks. For each set of alternative partner tasks, we also generate an abstract partner task representing all partner tasks in the set. Since the variability of customizable partner services are expressed as feature models with mapping to customizable capabilities, we also derive mapping models that represent the correspondence between alternative partner tasks and variant partner features, i.e. *PartnerTaskFeature mapping models.* Consequently, results of the service selection activity are a repository of (alternative) partner tasks and *PartnerTaskFeature mapping models*. It should be noted that in this methodology, the *PartnerTaskFeature mapping model* is intentionally generated before modeling the process.

In the third activity, the business process for the service composition is modeled using the extended metamodel. The identification of variation points and variants are based on the feature model identified in the first activity. Tasks from the partner task repository will be used to model the message flow between the service composition and partner services. The selection of (alternative) partner tasks and abstract partner tasks from the partner task repository will not only facilitate the reuse of variability provided by partner services in the process modeling, but also enable the use of already generated *PartnerTaskFeature mapping model* in capturing variability inter-dependencies. The result of this activity is a process model with variability.

In the next activity, the model mapping technique is exploited to first model variability intra-dependencies. That is, all variation points and variants in a process model are mapped to variant features in the feature model of the business process. The result is a *FeatureTask mapping model.* Since the *PartnerTaskFeature mapping model* is already produced, model transformation techniques are utilized to automatically generate *FeatureFeature mapping model* as mentioned.

The resulting software artifacts of the first four activities will be used in two different ways. Firstly, they are used for the derivation of process variants given a particular feature configuration as the result of a customization (i.e. *Variant Derivation* activity). Secondly, those software artifacts are used to generate the variability description of the resulting service composition (i.e. *Variability Description Generation* activity) which can contribute to other service compositions. Due to space limitation, we just describe the first usage in the following subsection.

6.2 Deriving Executable Process Variants

While the modeling of variability and variability dependencies is a design time process, the derivation of an executable process variant usually happens at runtime. This is triggered when the service composition is customized by consumers or the service provider itself. As explained, the customization is performed using the feature model of the service composition (cf. section 4.3) and generally requires the runtime customization of respective partner services.

Given a feature configuration of the composition, we exploit model transformation techniques as follows to derive a particular executable process variant:

1. The *FeatureTask mapping model* is referenced for specializing the process model. The process model is actually a model template which is the superset of all process variants. Therefore, those process elements, which are mapped to selected features, are maintained while those process elements, which are mapped to removed features, are purged from the process model. The result of this task is an abstract process variant which does not have variability but still contains partner tasks and consumer tasks. The detail of specializing a model template can be found in our previous work [5].
2. The *FeatureFeature mapping model* is referenced for generating a feature configuration for each customizable partner service. These feature configurations are used to customize corresponding partner services and produce particular partner service variants.
3. From the abstract process variant and partner service variants, an executable process variant is generated. We presume the use of BPEL for the executable process. It is important to note that the existence of partner tasks in the abstract process variant will help to create partner links and accurate service invocation between the process variant and partner service variants.
4. Finally, based on the information of consumer tasks in the abstract process variant, the service interface description of this process variant is generated.

At the end of this activity, a fully executable process variant that matches the given feature configuration is generated along with a service interface description. The process variant will invoke a set of automatically customized partner service variants.

7 Prototype Implementation

We have developed a prototype system for validating the feasibility of our approach. Key components are an Eclipse plugin for modeling business processes along with their variability (cf. Figure 7) and a model mapping tool for capturing all types of

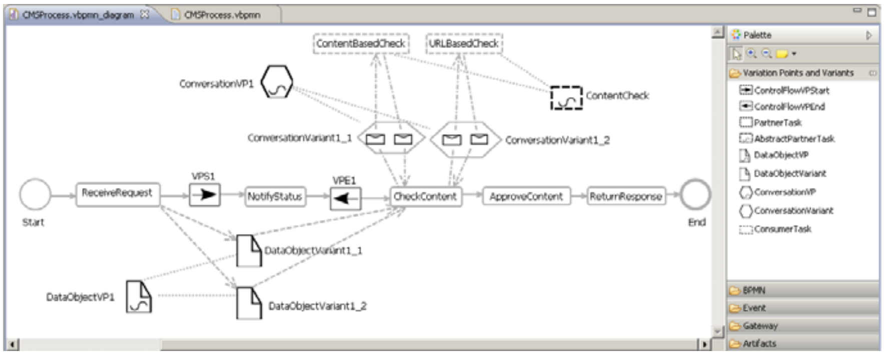
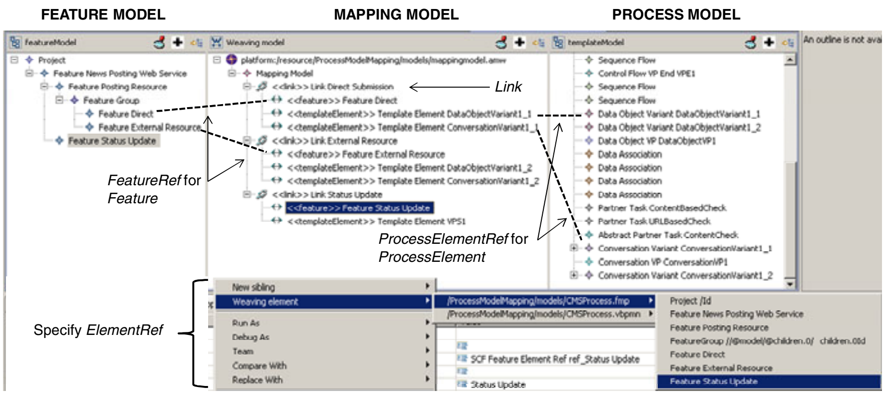


Fig. 7. A screenshot of modeling business processes with variability

variability dependencies (cf. Figure 8). Using our business process modeling tool, we successfully modeled the case study with all possible variation points and variants. This case study, despite of its simplicity, cannot be modeled by any approach in related work because all those approaches do not cater for the variability of partner services. In the following paragraphs, we introduce these key components.

Figure 7 is a screenshot of our process modeling tool. From the BPMN 2.0 metamodel, we extracted a subset that contains all model elements relevant to service compositions. We then introduced our process metamodel extension into the extracted metamodel. Our Eclipse plugin enables the development of any business process conforming to the extended metamodel. Modelers can select existing and new process elements from the right Palette tool. The screenshot displays the process model for the case study with three variation points: one variation point in the control flow, namely *VPS1* and *VPE1* for *ControlFlowVPStart* and *ControlFlowVPEnd*, one variation point in the data flow, namely *DataObjectVP1*, and one variation point in the message flow, namely *ConversationVP1*. *ConversationVP1* is associated with two alternative variants, namely *ConversationVariant1_1* and *ConversationVariant1_2*. While *ConversationVariant1_1* represents the message flow between the *"CheckContent"* task and the "*ContentBasedCheck"* partner task, *ConversationVariant1_2* represents the message flow between the same *"CheckContent"* task and the *"URLBasedCheck"* partner task. *"ContentBasedCheck"* and *"URLBasedCheck"* are alternative partner tasks associated with the same *AbstractPartnerTask*, namely *"ContentCheck"*. These *PartnerTasks*, *AbstractPartnerTask,* as well as the *PartnerTaskFeature* mapping model are generated from the service variability description of the ContentChecker service (cf. Figure 3). Modeling variability in this way enables the capturing of variability inter-dependencies between the CMS Provider and the ContentChecker service.

Figure 8 is a screenshot depicting how to capture variability intra-dependencies, i.e. *FeatureTask* mapping model, using our model mapping tool. In the screenshot, the feature model is presented in the left panel, while the process model is presented in the right panel and the middle panel presents the mapping model. Three mapping links are created in the screenshot. The first link associates the "Direct" feature with one *DataObjectVariant* and one *ConversationVariant*. Similarly, the second link associates the "External Resource" feature also with one *DataObjectVariant* and one *ConversationVariant*. The third link associates the "Status Update" feature with one *ControlFlowVP*. The creation of model elements is based on the context menus as shown in Figure 8. This component is implemented as an extension to Atlas Model Weaver (AMW) [24]. We then perform model transformations using Atlas Tranformation Language (ATL) [25] to derive *FeatureFeature* mapping model. As explained, *PartnerTaskFeature* mapping model and *FeatureFeature* mapping model account for variability inter-dependencies.



```
featureModel
Project
  Feature News Posting Web Service
    Feature Posting Resource
      Feature Group
        Feature Direct
        Feature External Resource
    Feature Status Update
```

```
Weaving model
platform:/resource/ProcessModelMapping/models/mappingmodel.amw
  Mapping Model
    <<link>> Link Direct Submission
      <<feature>> Feature Direct
      <<templateElement>> Template Element DataObjectVariant1_1
      <<templateElement>> Template Element ConversationVariant1_1
    <<link>> Link External Resource
      <<feature>> Feature External Resource
      <<templateElement>> Template Element DataObjectVariant1_2
      <<templateElement>> Template Element ConversationVariant1_2
    <<link>> Link Status Update
      <<feature>> Feature Status Update
      <<templateElement>> Template Element VPS1
```

```
templateModel
  Sequence Flow
  Control Flow VP End VPE1
  Sequence Flow
  Sequence Flow
  Data Object Variant DataObjectVariant1_1
  Data Object Variant DataObjectVariant1_2
  Data Object VP DataObjectVP1
  Data Association
  Data Association
  Data Association
  Data Association
  Partner Task ContentBasedCheck
  Partner Task URLBasedCheck
  Abstract Partner Task ContentCheck
+ Conversation Variant ConversationVariant1_1
  Conversation VP ConversationVP1
+ Conversation Variant ConversationVariant1_2
```

```
/ProcessModelMapping/models/CMSProcess.fmp
/ProcessModelMapping/models/CMSProcess.vbpmn

SCF Feature Element Ref ref_Status Update
Status Update

Project /Id
Feature News Posting Web Service
Feature Posting Resource
FeatureGroup //@model/@children.0/   children.0ld
Feature Direct
Feature External Resource
Feature Status Update
```

Fig. 8. A screenshot of a mapping model between a feature model and a process model

8 Conclusion

In this paper, we have proposed a feature-oriented approach to modeling and managing variability in process-based service compositions. We have extended the BPMN 2.0 metamodel for introducing variation points and variants in all three aspects of service compositions, i.e. control flow, data flow, and message flow. These extensions enable not only comprehensive modeling of variability, but also the generation of executable process variants as the result of a service customization. In addition, we have introduced a feature mapping technique for capturing not only variability intra-dependencies among variants within a process model, but also variability inter-dependencies between variants in a process model and variants in partner services. Consequently, our approach is able to address the situation when a customizable composition is orchestrated using partner services some of which are customizable. This is not achievable using existing approaches.

We have also described a methodology that facilitates the development of business processes conforming to the extended process metamodel with systematic variability management. The key advantage of the methodology is the systematic exploitation of variabilities provided by partner services to increase the chance of reusing variability. The methodology exploits MDE techniques for automating most parts, especially the generation of executable process variants. In addition, we present a prototype system for demonstrating the feasibility of our approach.

As future work, we plan to develop techniques for the generation of service variability description from the process model leading to a framework for the recursive delivery of customizable services in service ecosystems.

Acknowledgments. This research was carried out as part of the activities of, and funded by, the Smart Services Cooperative Research Centre (CRC) through the Australian Government's CRC Programme (Department of Innovation, Industry, Science and Research).

References

1. Sun, C.-A., et al.: Modeling and managing the variability of Web service-based systems. Syst. & Softw. 83(3), 502–516 (2009)
2. Hallerbach, A., et al.: Capturing variability in business process models: the Provop approach. Softw. Maint. & Evol.: Res. & Pract. 22(6-7), 519–546 (2010)
3. Svahnberg, M., et al.: A taxonomy of variability realization techniques: Research Articles. Softw. Pract. Exper. 35(8), 705–754 (2005)
4. Barros, A.P., et al.: The Rise of Web Service Ecosystems. IT Prof. 8(5), 31–37 (2006)
5. Nguyen, T., et al.: A Feature-Oriented Approach for Web Service Customization. In: IEEE Int. Conf. on Web Services, pp. 393–400 (2010)
6. Stollberg, M., Muth, M.: Service Customization by Variability Modeling. In: Dan, A., Gittler, F., Toumani, F. (eds.) ICSOC/ServiceWave 2009. LNCS, vol. 6275, pp. 425–434. Springer, Heidelberg (2010)
7. Liang, H., et al.: A Policy Framework for Collaborative Web Service Customization. In: Proc. of the 2nd IEEE Int. Sym. on Service-Oriented System Engineering (2006)
8. Schmid, K., et al.: A customizable approach to full lifecycle variability management. Science of Computer Programming 53(3), 259–284 (2004)
9. Nguyen, T., et al.: Managing service variability: state of the art and open issues. In: Proc. of the 5th Int. Workshop on Variability Modeling of Software-Intensive Systems (2011)
10. Pohl, K., et al.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer-Verlag New York, Inc. (2005)
11. Hadaytullah, et al.: Using Model Customization for Variability Management in Service Compositions. In: IEEE Int. Conf. on Web Services 2009 (2009)
12. Koning, M., et al.: VxBPEL: Supporting variability for Web services in BPEL. Information and Software Technology 51(2), 258–269 (2009)
13. Mietzner, R., et al.: Generation of BPEL Customization Processes for SaaS Applications from Variability Descriptors. In: IEEE Int. Conf. on Services Computing 2008 (2008)
14. Razavian, M., et al.: Modeling Variability in Business Process Models Using UML. In: 5th Int. Conf. on Information Technology: New Generations 2008 (2008)
15. Chang, S.H., et al.: A Variability Modeling Method for Adaptable Services in Service-Oriented Computing. In: Proc. of the 11th Int. Conf. on Software Product Line (2007)
16. Schnieders, A., et al.: Variability Mechanisms in E-Business Process Families. In: Proc. of Int. Conf. on Business Information Systems, pp. 583–601 (2006)
17. Kang, K.C., et al.: Feature-Oriented Domain Analysis (FODA) Feasibility Study, in Technical Report, Softw. Eng. Inst., CMU. p. 161 pages (November 1990)
18. Kang, K.C., et al.: FORM: A feature-oriented reuse method with domain-specific reference architectures. Ann. Softw. Eng. 5, 143–168 (1998)
19. Batory, D.: Feature Models, Grammars, and Propositional Formulas. In: Obbink, H., Pohl, K. (eds.) SPLC 2005. LNCS, vol. 3714, pp. 7–20. Springer, Heidelberg (2005)
20. Griss, M.L., et al.: Integrating Feature Modeling with the RSEB. In: Proc. of the 5th Int. Conf. on Software Reuse (1998)
21. Czarnecki, K., et al.: Formalizing cardinality-based feature models and their specialization. Software Process: Improvement and Practice 10(1), 7–29 (2005)
22. Sinnema, M., Deelstra, S., Nijhuis, J., Dannenberg, R.B.: COVAMOF: A Framework for Modeling Variability in Software Product Families. In: Nord, R.L. (ed.) SPLC 2004. LNCS, vol. 3154, pp. 197–213. Springer, Heidelberg (2004)

23. Benavides, D., et al.: Automated analysis of feature models 20 years later: A literature review. Information Systems 35(6), 615–636 (2010)
24. Didonet, M., et al.: Weaving Models with the Eclipse AMW plugin. In: Proceedings of Eclipse Modeling Symposium, Eclipse Summit Europe (2006)
25. Jouault, F., et al.: ATL: A model transformation tool. Science of Computer Programming 72(1-2), 31–39 (2008)