Feature Modeling for Service Variability Management in Service-Oriented Architectures

Mohammad Abu-Matar, Hassan Gomaa, Minseong Kim, and Ahmed Elkhodary
George Mason University
{mabumata, hgomaa, mkim12, aelkhoda}@gmu.edu

Abstract – Service Oriented Architecture (SOA) has emerged as a model for distributed software development that promotes flexible deployment and reuse. Software product lines (SPL) promote reusable application development for product families. Service oriented systems change to respond to changing clients’ requirements. As they change, service oriented systems can be modeled as service families similar to the SPL concept. Some SPL development techniques rely on feature models to describe the commonality and variability of member applications. We use SPL feature modeling techniques to model the variability of service families. In this paper, we introduce a platform independent approach to model SOA variability based on feature modeling. We develop a UML meta-model to describe features. Then, we describe SOA variability scenarios using the newly released OMG standard SoaML. Finally, we develop a meta-model that maps features into SoaML. We believe that such an approach facilitates variability management of service families in a systematic and platform independent way.

1. Introduction

Software Product Lines (SPL) are families of software systems that share common functionality, where each member has variable functionality [1]. The main goal of SPL is the rapid development of member systems by using reusable assets from all phases of the development life cycle. This goal is similar to the goal of Service Oriented Architecture (SOA) where flexible application development is a common theme.

An essential modeling phase in Software Product Lines Engineering (SPLE) is Commonality and Variability Analysis (CVA) where the common and varying features of SPL member applications are outlined. CVA is commonly expressed in Feature Models based on the SPL common, optional, and alternative use cases.

Since services in SOA could be used by different clients with varying functionally, we believe that SOA variability modeling can benefit from SPL variability modeling techniques. Service oriented systems can behave like service families, similar to the concept of software product lines.

However, variability modeling in SOA has different challenges to deal with than non-SOA software product lines, because service consumers are decoupled from services providers and application development is usually done by assembling services rather than developing components and code. Therefore, the following two aspects of variability in SOA must be thoroughly taken into account:

- Variability of consumer services must remain independent from the provider services.
- Service oriented collaboration of multiple enterprises happens in a decentralized and federated manner. This federation, also called choreography, creates a very loose coupling with no central point of authority. Thus, variability of federated enterprises has to be taken into account as well.

Existing approaches to handling variability in SOA [6, 15, 16, 17] have used SPL concepts to model variability in service families (this will be discussed in detail in the Related Work section), however none has produced a treatment of all the variability aspects in SOA mentioned above. In addition, existing research addresses centralized variability without considering choreography, which is decentralized collaboration. In particular, existing research mainly treats SOA variability issues in a platform specific way by focusing on Web Services and orchestration languages like Business Process Execution Language (BPEL).

In this paper, we introduce an approach that addresses all variability aspects mentioned above in a unified manner. In particular, our approach handles decentralized as well as centralized variability scenarios in a platform independent way.

In doing so, we follow a platform independent approach by using the newly released OMG standard SoaML [11]. SoaML is a UML extension released by the Object management Group (OMG). SoaML is “a standard way to architect and model SOA solutions using UML” [11]. SoaML introduces new elements that model SOA concepts based on existing UML elements by using the Profile extension mechanism in UML. In this paper, we only use the SoaML modeling elements that are relevant to our current research.
Thus, the **key concepts of our approach** are as follows, in which we combine SPL variability modeling [1] concepts with SOA concepts, as represented in SoaML:

1. Feature modeling – where we use SPL’s feature modeling techniques to model variability on a meta-modeling level for services.
2. Choreography variability – where we exploit UML and SoaML’s support for Contracts and SPL feature and variability modeling to model the variability of collaborating entities in decentralized environments.
3. Orchestration variability – where we exploit feature modeling to model the variability of service coordination and relate this to SoaML support for workflows.
4. Interface Variability – where we exploit SPL feature modeling and variable interface modeling [1] to model the variability of SoaML service interfaces.
5. Implementation Variability – where we use SPL feature modeling and class modeling variability [1] to model the variability of SoaML service implementations.

We use UML extension mechanism (UML Profiles) to model feature models based on our previous work [12]. Then, we describe SOA variability scenarios using the newly released OMG standard SoaML. Finally, we develop a meta-model that maps SPL feature and variability modeling to SoaML.

The rest of the paper is structured as follows. In section 2, we present a motivating example. Feature oriented service variability, and the feature to service mapping meta-model are discussed in section 3 and 4. We detail related work in section 5, and conclude the paper in section 6.

2. **Motivation**

In a typical SOA environment, each community has a central body or federation in charge of defining business contracts and policies. This process encompasses defining federation-level business contracts as well as organization-level services. For example, the E-Commerce B2B Network (EBN) in Figure 1, a central authority typically defines business-to-business contracts (**Service Contracts**) and business services (**Service Interfaces**) required in order to conduct commercial activities. Each **Service Contract** prescribes generic roles for the organizations participating in it (**Participants**).

For example, in Figure 1a, an organization that aims to play the **Seller** role in the **Purchasing** supply chain must be able to implement and advertise the service interface **Ordering Service**. In this way, each organization defines its own business processes while satisfying federation-wide business contracts.

The variability of the **EBN** can take place at either the federation-level or the organization-level. This can be manifested by introducing new Service Contracts, Service Interfaces, and/or Participants. We address the following four variability scenarios:

- Service Contract (Choreography) Variability - Some contracts may be introduced to the architecture to satisfy the needs of particular realizations of the supply chain. For example, in Figure 1a, the Credit Checking service...
contract is introduced, if EBN is operating within a zone of non-trusted buyers and sellers.

- **Orchestration Variability** – A business process is composed of a sequence of activities, and this sequence is called an orchestration. Participants may need to change their internal business processes without affecting other participants in the contract. For instance, a Seller in EBN may be able to provide some discounts to its buyers to beat the competition. To do so, the Order Fulfillment orchestration, as shown in Figure 1b, must introduce a new Calculate Discount activity to the workflow.

- **Participants Interface Variability** – New service interfaces may be introduced to a participant so that it can be involved in new choreographies. For example, by introducing a CheckCreditRating service interface in Figure 1c, a Seller can participate in a Credit Checking contract in Figure 1a.

- **Service Implementation Variability** – An application can be configured to have one or more payment options. In EBN for example, the Seller’s infrastructure can allow more than one payment option as given by the Make Payment service, in Figure 1d, which is specialized to allow Direct Deposit or Credit Card implementation logic. A given Seller may be configured to support either or both of these options, with Credit Card as default.

3. **Feature Oriented Service Variability**

In the proposed approach, through feature modeling, we can analyze and model commonality and variability in service families with respect to the variability of services.

Feature modeling is rooted in the seminal work of Kang et al. [5] in the Feature Oriented Domain Analysis (FODA) method. It is the activity of identifying externally visible characteristics of products in terms of features and organizing them into a feature model. In a software product line approach, feature models are used to express and manage similarities and differences among different family members in a product line. Features are analyzed and categorized as common, optional, or alternative. Common features among products in a product line are mandatory or kernel features, while different features among them may be optional and alternative. Related features can be grouped into feature groups, which constrain how features are used by a product of a product line. A feature group is specialized to “zero-or-more-of”, “at-least-one-of”, “exactly-one-of”, or “zero-or-one-of” [1].

The commonalities and differences between members in a service family are therefore expressed in terms of features. A feature is a reusable requirement that can be selected by any member of the service family.

Referring to our running example, after initial analysis, it was determined that the B2B Network could vary in the following ways:

- The B2B Network could have a credit checking service contract for regular customers, which would then require checking credit rating; however, if the business always deals with trusted buyers, this capability is not required. This requirement can be modeled as a ‘Check Credit Rating’ <<optional feature>>.

- The Order Fulfillment Process could offer two types of payments: credit card, which is default, and electronic check. This requirement can be modeled as a ‘Payment’ <<at least-one-of feature group>> with a ‘Credit card’, which is a default feature stereotyped with <<default feature>>, and an ‘Electronic check’, which is an optional feature, <<optional feature>> in the feature model.

- The Seller Participant could offer a discount capability that can be selected seasonally. This requirement can be modeled as a “Discount” feature, which has the <<optional feature>> stereotype.

- The Order Fulfillment process could offer a ‘Preferred Customer’ capability for customers with existing credit records to speed up their order processing. This requirement can be modeled as a ‘Consumer Type’ <<at least-one-of feature group>> with a ‘Preferred Customer’ <<alternative feature>> and a ‘Regular Customer’ default feature in the feature model.

Based on the aforementioned variability scenarios, we create the following feature model:

![Feature Model Diagram]

**Figure 2: The E-Commerce Feature Model**

The following sub-sections describe how feature-oriented service variability has been achieved in SoaML via meta-modeling.
3.1 SoaML Meta-Model

Meta-modeling is a way to define models [3]. A meta-model defines what elements can exist in a modeling language, e.g., the UML meta-model defines concepts like Classes, Packages, and Associations [2]. In addition, the meta-model describes the relations among the modeling elements in the language. All elements in a modeling language are defined in a meta-model for that language.

Before we describe variability in the SoaML meta-model, we briefly define the relevant SoaML elements that have been extended to support variability:

- **ServiceArchitecture** – Specifies a community of Participants who collaborate together to achieve some goals. This element extends the UML Collaboration element.
- **ServiceContract** – Specifies the agreement between providers and consumers, and may include interfaces, policies and role descriptions. This element extends the UML Collaboration element.
- **ServiceInterface** – Models the service interface concept in SOA. It specifies provided and required interfaces by participants in addition to the interaction protocol that describes how provided and required interfaces should be called. This element extends the UML Class element.
- **Participant** – Specifies providers or consumers of services. This element extends the UML Component and Class elements.

3.2 Variability Meta-Model

Since neither UML nor SoaML has native support for variability modeling, we use a UML based feature meta-model based on [12]. The variability meta-model (Figure 3) specifies how the different meta-classes relate to each other. The meta-classes modeled include service related meta-classes such as service contract, service interface, and participant, as well as SPL meta-classes, in particular the feature meta-class. Each of these meta-classes is specialized using SPL concepts into kernel, optional, and variant meta-classes. The top right side of Figure 3 below depicts a feature meta-model. Features are specialized into kernel, optional, alternative, and default depending on the characteristics of the requirements, that is, commonality and variability.

Kernel features are requirements common to all members of systems, that is, required by all members of a product line. Optional features are required by only some members of a product line.

An alternative feature is an alternative of a kernel or optional feature to meet a specific requirement of some systems. Default features are chosen by default when they are part of a feature group. Feature groups refer to constraints on the selection of a group of features (e.g., preventing selection of mutually exclusive features). Feature dependencies represent relationships between features.

In the following section, we map feature models into SoaML models at the meta-level to cope with variability in service families.

4. Feature to Service Mapping

In this section, we provide meta-modeling mapping between a feature model and SoaML. In other words, we establish mapping between feature meta-model elements and the relevant SoaML meta-model elements based on the service variability scenarios discussed in the previous section.

4.1 Service Contract Feature Mapping

In Figure 3, a Feature maps to a set of ServiceContracts. For example, when feature Check Credit Rating is selected, service contract Credit Checking will be activated and enforced in the community. The variability stereotype on a ServiceContract dictates the type of feature it may map to. For instance, an optional feature (e.g., Check Credit Rating) can only contain optional service contracts (e.g., Credit Checking service contract). Similarly, an alternative feature may contain Variant service contracts only).

4.2 Service Orchestration Feature Mapping

In Figure 3, Feature maps to a set of Activities in the Participant’s orchestration. For example, when the Discount optional feature is selected in EBN, which means that the system changes to provide the ‘Discount’ capability, the Calculate Discount activity becomes part of the Order Fulfillment process. Thus, the Discount <<optional feature>> is mapped to <<optional>> Calculate Discount activity.

4.3 Service Interface Feature Mapping

In Figure 3, a Feature maps to a set of ServiceInterfaces. For example, if the Check Credit Rating optional feature is selected, the Seller participant has to provide a new interface that can interact with a credit rating agency. Thus, the Check Credit Rating <<optional feature>> is mapped to <<optional>> Check Credit Rating ServiceInterface.
4.4 Service Implementation Feature Mapping

In Figure 3, a Feature maps to a set of service implementations. In the running example, if the Electronic Check alternative feature is selected, the Payment ServiceInterface would have a different implementation. Thus, the Electronic Check <<alternative feature>> is mapped to <<alternative>> Electronic Check ServiceImplementation.

5. Related Work

There have been several approaches for modeling variability in service oriented architectures. The authors in [13] add variability analysis techniques to an existing service oriented analysis and design method (SOAD). Decision tables are used in [13] to record variability types in each phase of the SOAD process. Finally, these tables are used to map business process requirements into service components.

The authors in [14] present architectural pattern approaches to model variation points in Web Services. They point out that the standards used for Web Services have inherent support for variability.

The authors in [15] advocate a lightweight product line engineering approach that has a specific phase for service composition in the product line architecture. They introduce several variation points that can be used to customize the product line during service selection in the application engineering phase. However, the authors in [15] do not tie service selection to the features required in the product line.

The authors in [16] presented a software product line engineering approach based on Web Services. Components in the architecture were modeled using the <<web service>> stereotype to indicate the use of Web Services. UI feature selection determined the selection of Web Services. UML activity diagrams modeled customization choices based on feature/Web Service interactions.

In [6], the authors used the concept of features to solve variability problems for SOA. The authors used Feature Oriented Programming techniques to modify the base code of services based on requested features. However, the authors’ approach assumes the availability of service implementation code, which is not the norm in true SOA scenarios.

The authors in [17] proposed a method to managing variations in SOA systems and keeping services reusable and useful over a long period of time by adapting a feature-oriented product line approach. Yet, they do not consider the relationships between features and services especially with respect to the SOA variability scenarios.

6. Conclusion/Discussion

One of the major benefits claimed for SOA is the flexible building of IT solutions that can react to changing business requirements quickly and economically. SOA promises a vision where service providers offer their services based on their expertise and service requesters search and discover these services based on their business needs.

Normally, service providers are decoupled from service requesters, thus requesters and providers change independent of each other. As a result, variability in SOA is more challenging to deal with than in traditional software systems. In addition, the service oriented choreography of multiple enterprises happens in a decentralized manner. This federation, also called choreography, creates a very loose coupling with no central point of authority. Thus, variability of federated enterprises has to be taken into account as well.
In this paper, we introduced an approach that addresses service oriented variability aspects in a unified manner. In particular, our approach aims to handle decentralized as well as centralized variability in a platform independent way. We used software product line feature modeling techniques and SoaML to model SOA variability at a metamodeling level. Especially, we described the SOA variability scenarios using SoaML.

Finally, we developed a meta-model that maps features to SoaML meta-classes. It should be pointed out that although we described our SPL approach to service features and variability modeling with SoaML meta-classes, our approach could also be used with other service metamodeling approaches.

We believe that our approach to service feature and variability modeling has several benefits:

- The use of established SPL feature modeling to manage variability in service families.
- Extended SoaML with variability modeling.
- A service could participate in many service family members, thus maximizing reusability.
- Service providers and consumers can change independent of each other.

In our ongoing research, we plan to achieve the following goals:

- Add Participant variability to our aforementioned service variability types.
- Add more details to the Feature-to-Service meta-model mapping coupled with OCL consistency rules.
- Create transformation rules that transform features in feature models to service artifacts in SOA environments. By following an MDA approach, our variable service model represents the Platform Independent Model (PIM). In essence, we will have two kinds of transformation definitions. The first transformation definition would create the service member application from the platform independent SOA model. The second transformation definition would transform the derived service member application into an SOA platform specific model (PSM).
- Tool support to manifest our approach.

7. References