A Variability Modeling Method for Adaptable Services in Service-Oriented Computing

Soo Ho Chang and Soo Dong Kim
Department of Computer Science
Soongsil University, Seoul, Korea
shchang@otlab.ssu.ac.kr    sdkim@ssu.ac.kr

Abstract
Publish-Discover-Compose paradigm of Service-Oriented Computing (SOC) presents a challenge on service applicability. Services are not just for predefined clients, rather for potentially many unknown clients. Hence, published services should be highly adaptable to various service clients and contexts. For that, service variability must carefully be modeled by considering the unique computing paradigm and requirements of SOC such as dynamic discovery and composition of services. Current SOC approaches to modeling services largely focus on defining business processes and service components without considering service variability in sufficient details. In this paper, we first compare the variability on conventional applications and the variability on SOC. Then, we identify four types of variability which may occur on services. As the main contribution of our research, we present a method to model service variability and design adaptable services for each type of service variability. Using the proposed framework, we believe the applicability and reusability of such services can be greatly increased.

1. Introduction
In Service-Oriented Computing (SOC), new services are published in service registry, discovered by service clients, composed into the target service which fulfills the service requested. Service-Oriented Architecture (SOA) is a particular form of SOC for providing services on web platform, called Web Services. The paradigm of Publish-Discover-Compose of SOC presents a challenge on service applicability and adaptability.

Since services are not just for predefined clients, rather for potentially many unknown clients, they have to be highly adaptable to various service clients and contexts. To leverage the service reusability and adaptability, the variability presented on services should carefully be modeled by considering the unique computing paradigm and requirements of SOC such as dynamic discovery and composition of services.

Current SOC approaches to modeling services largely focus on defining business processes and service components without considering service variability in sufficient details. Through our research and experience, we believe that product-line engineering (PLE) has a great potential in modeling service variability and developing high adaptable service components.

In this paper, we present a technical comparison between the variability on conventional applications and the variability on SOC. Then, we identify four types of variability which may occur on services. As the main contribution of our research, we present a method to model service variability and design adaptable services for each type of service variability. Using the proposed framework, we believe the applicability and reusability of such services can be greatly increased.

2. Related Works
2.1. On the Variability Analysis and Design in PLE
Feature-Oriented Domain Analysis (FODA) proposes features as the unit to represent system characteristics [1]. The features are represented with three kinds of relationships: mandatory, alternative, and optional. KobrA is another approach to component-based product line engineering from Fraunhofer IESE [2]. This work also defines variability with variation points and variants. For representation of variability, this work proposes UML stereotypes for analysis and design model. COVAMOF is a variability modeling approach that represents variation points and dependencies and supports the modeling of relations between dependencies [3]. Pohl’s work is one of the works emphasizing the variability issues in domain engineering and application engineering [4]. Kim’s work define three kinds of relationships; Binary, Selection, and Open [5].
To model variability, most works includes variation points and variants where the types of variation points are originated from alternative, optional, and open. There are representations for the variability such as feature model, UML stereotypes, decision model, and so on. However, these types and notations are for general software domains. For SOA, modeling methods specific to SOA should be proposed.

2.2. On the Service Adaptation

Sam’s work proposes a framework for dynamic Web services customization [6]. To adapt service compositions, this work defines structures and constraints for Web services. Using the structures, matchmaking process is conducted with two types of comparisons; syntactic/semantic comparison of context, input, and output, and constraint comparison of the input and output. The work also identifies conflict problems of the input and output types between requested services and available services. And, it proposes a mechanism for customizing advertised services to be compatible with the requested services. However, the conflict type defined in this work is limited to interfaces.

Kongdenha’s work proposes an aspect-oriented framework for service adaptation [7]. The framework proposes taxonomy of potential mismatch types between external specifications and service implementations. And it includes a repository of aspect-based templates to automate the task of handling mismatches. The mismatch types identified here are signature mismatch between input types and ordering mismatch between flows of messages. To be practical, this can be further extended with methods for design.

Jiang’s work addresses the notion of reuse in Web service development, i.e., families of Web services [8]. The work proposes a categorization of possible variation points in Web services, and introduces a pattern-based approach for managing the variation points and specifying a Web service framework. The variation points are defined on endpoints, WSDL descriptions, and business logic. However, specific methods to identify and to handle the variation are not yet defined.

There exist others adaptation approaches such as [9] and [10]. In summary, current service adaptation approaches focus on adapting business processes or service compositions. And, the design methods proposed are limited to identification of mismatch and interface mapping, rather than covering the whole scope of variability which can occur on SOC.

3. Variability in PLE and SOC

PLE provides good methods to model variability in a family. Due to its difference from SOC paradigm, however, the key methods of PLE should be extended. In this section, we identify the difference between PLE and SOC in terms of reuse approaches. Then, we identify the elements of PLE variability modeling which can be extended to support variability modeling for SOC.

3.1. Comparing Reuse in PLE and SOC

Core assets are the reusable units in PLE where they embed common and variable features in a product family. Services are the reusable units in SoC where they provide functionality of several clients and they can be adapted to the clients.

As reusable units, the core assets and the services have slightly different characteristics as shown in the Table 1.

Table 1. Comparison Table

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Reuse in PLE</th>
<th>Reuse in SOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consumer</td>
<td>Application</td>
<td>Service Caller</td>
</tr>
<tr>
<td></td>
<td>Developer</td>
<td></td>
</tr>
<tr>
<td>Scope of Consumers</td>
<td>Family Members</td>
<td>Many (un)known</td>
</tr>
<tr>
<td></td>
<td>known</td>
<td>Clients</td>
</tr>
<tr>
<td>Reuse Unit</td>
<td>Core asset</td>
<td>Service</td>
</tr>
<tr>
<td>Results of Reuse</td>
<td>Generating</td>
<td>Delivering Services</td>
</tr>
<tr>
<td></td>
<td>Applications</td>
<td></td>
</tr>
<tr>
<td>Coupling with Reuse Assets</td>
<td>Tightly Coupled</td>
<td>Loosely Coupled</td>
</tr>
<tr>
<td>Reuse Method</td>
<td>Instantiation</td>
<td>Service Invocation (including adaptation)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time to be Reused</td>
<td>Static</td>
<td>Static and Dynamic</td>
</tr>
<tr>
<td>Payment Type</td>
<td>Pay for Own</td>
<td>Pay for Use</td>
</tr>
</tbody>
</table>

Core assets are reused by application developers where the application is a member of the product family. But, the services are invoked by service clients such as user interfaces and other larger-grained services orchestrators or choreographers [11]. And the members in PLE tend to be known while there are many unknown the clients of the services.

Core assets in PLE including a generic architecture and components are used to develop applications, whereas services are delivered in SOC. Therefore, core assets and applications are tightly coupled in PLE while services are loosely coupled with service clients.

For the reuse time, core assets are used in application development time which is static, while service invocations are decided at design time and can be reconfigured at run time.
For the payment type, the consumer of core assets pays to own the core assets, whereas the service consumer pays to use (i.e. invoke) the services. In summary, conventional PLE approaches should and can be well extended to model the service variability.

3.2. General Elements of Variability Model

Although there are differences with SOC, there are elements of variability model in PLE. In general, the variability can be represented with variation point, variant, types of variation points, and dependency.

A variation point is a place in software where the minor difference occurs [5]. Variant is a value or instance that can validly fill in a variation point. That is, a variant resolves a variation point. The variable features are classified to alternative and optional features. Moreover, some features can be expected to be variable in the future called open.

In the terms of analyzing and modeling domain, variability modeling approaches in SOA and PLE are similar. Hence, the elements of PLE variability can be used to represent the variability in SOA. However, there are differences on representative artifacts. For example, analysis and design models are generally represented in UML. However, a business process for a service is generally specified business modeling languages such as [12] or [13].

Therefore, identifying representative artifacts in SOA is essential and the traceability among the representation should be seamless.

4. A Taxonomy of Variability in SOA

4.1. Key Artifacts of SOAD

SOA-based applications typically have a layered architecture, and we present a four-layered SOA architecture in terms of abstraction level.

**Business Process Layer:** As the top layer, Business Process Layer is to define business processes expected by service clients. Business Process (BP) represents a cohesive unit of the service perceived by service clients, not by component engineers. Typically, a BP is a larger-grained than a use case and a method of objects, and it is defined with a service workflow among smaller-grained activities. Hence, Business Process Specification includes workflows of participating activities. For representation of business process, business modeling language [11] and business process execution language [13] are generally used.

**Unit Service Layer:** Activities of a business process are conceptual units of works, perceived by clients. It will eventually be performed by a software element, which we call a unit service, UnitService. That is, an activity is fulfilled by running a unit service. The main distinction between them is that an activity is a conceptual unit perceived by clients and a UnitService is its corresponding task defined from engineering perspective. Hence, the notion of UnitService is a vehicle that bridges clients’ view to engineers’ view. Some unit services may be common among the business processes, and hence they are reusable among several business processes.

**Service Interface Layer:** In SOC, the interfaces of services are specified separately from service components, and service providers publish the services in WSDL [14] in UDDI service registries. Hence, the unit services identified should be bound to interfaces of the published services which fulfill the requirement of the unit services. Therefore, the Service Interface Layer contains the interfaces of services published by service provides, and it separates the unit services from the service components. By having this layer, unit services can be bound to any compatible interfaces, and the interfaces can be realized by and bound to any compatible service components.

**Service Component Layer:** This layer is to specify service components which implement the service interfaces. Some components are like the one in component-based development (CBD), and typically implemented with objects on OO/CBD platforms such as EJB. Other components can be simply wrappers of legacy applications. Although there is a difference between the two types on how components are implemented, but they both have to provide physical interfaces that conform to the published interfaces (in WSDL) of the Service Interface Layer.

4.2. Types of Variation Points

Service variability is rooted to the slightly different service requirements of clients and their context. Hence, the variability is identified by analyzing the service requirement, more specifically business processes and unit services.

From the variability information of business processes and unit services, we identify four types of service variability; Workflow, Composition, Interface, and Logic Variability, as shown in Figure 1.

**Workflow Variability:** A business process is carried out in a sequence of unit services, called workflow. In a workflow, some unit service may not be requested to a particular service user. And some part of the sequence may be differently performed to other particular service user. We now define Workflow Variability for
the variation on the workflow of a business process. That is, unit services can be alternatively or optionally fulfilled in a workflow depending on the individual service users.

Variability presented in Business Process and Unit Service

**Workflow Variability**
- Different Web services for a unit service according to NFR rather than FR
- The interfaces of candidate web services are different

**Composition Variability**
- The interfaces of candidate web services are different

**Interface Variability**
- The interfaces of candidate web services are different

**Logic Variability**
- The interfaces of candidate web services are different

Figure 1. Four Type of Variability

**Composition Variability**: A business process composes several unit services to fulfill the end users’ requirements. For one unit service in the workflow, there may be more than one possible service interfaces which implement the service with different implementation logics or quality attributes. In this case, variation occurs on selecting the most appropriate service interface. That is, depending on user requirements, different service interfaces of the implementation can be composed and we call it Composition Variability.

**Interface Variability**: It occurs when the interfaces of unit services do not match to the interfaces of services published in UDDI service registry. Recall that the unit service is a logical unit of service that is composed into various business processes. The services registered in a service registry are published and available services, of which interfaces are typically derived from implemented service components and interface specifications of some standards.

**Logic Variability**: A service component includes operations for providing the functionality of unit services. Then, we define business logic as an algorithm or logical procedure used in the operations. The service component should provide different logics depending on particular requested services. It is called Logic Variability.

Figure 2 shows the key artifacts of SOAD and possible variation points in this section.

5. A Variability Modeling Method

The SOA artifacts representing workflows, composition, and interfaces are different from those for conventional software. However, the logics are same as the logic in the component models. Therefore, in this section we propose the representations for the three types of variability; workflow, composition, and interface variability.

The artifacts in SOA are based on XML which means that it can be extended with XML scheme. For variability modeling in SOA artifacts, we define an xml scheme of variability and use the scheme in the corresponding artifacts as shown in the Figure 3.

![Figure 3. A Four-Layered SOA Architecture](image)

There are two representative artifacts; BPEL for business process modeling and WSDL for specifying service interfaces. The artifacts include workflow and composition variability and interface variability respectively. In this section, we represent the variability scheme and its application in the artifacts

5.1. VType and Decision Model

Drawing from the PLE research on variability, we define an element, *Variability*, which includes five key elements and attributes of the variability, called *VType*, as shown in the Figure 4.
Figure 4. A Scheme of Variability in XML.

The vp is to represent variation points which are classified into four types; workflow, composition, interface, and logic. Depending on the location where this tag is represented, applicable types are different. For example, workflow and composition types can be represented in business process models.

The scope is represented with three types; alternative, optional, and open. It means that variants of the vp can be alternatively or optionally selected. Or it can be further defined.

The variant is the value which will be chosen when its condition is satisfied. The condition is derived from the requirements of clients to which the business process should adapt.

The dependency is to represent relationships among several instances of the VType. For example, composition variability generally introduces interface variability in dynamic composition.

Based on the scheme, we define a decision model for a business process which is specified as a XML document. The decision model provides attached tasks for each variant in a variation point, where some tasks are referred as adaptation methods in service compositions at run time.

5.2. Workflow and Composition Variability

The business process includes two types of variability; workflow and composition variability. For the variability in a business process, we use the extension mechanism provided in BPEL[13].

In a business process, workflow variability is represented in the unit services; optional and alternative unit services.

The composition variability uses same representation to the workflow variability except for the different values of vpType. However, the attached tasks in decision model should include composition rules and activities so that the variable services can be dynamically composed depending on the run time context.

5.3. Interface Variability

Since interface variability occurs between required interfaces of a business process, the variability model present on the relationship. That is, the required interface of a business process can be variation point and the provided interface of candidate service components can be variants of the variation point as shown in the Figure 6.

5.4. Usages of the Variability Models

Variability modeling in SOA artifacts is in very early stage although it is one of essential prerequisites to provide adaptable services. With the variability model, we represent benefits and applications.
First, the models can be representation of variability analysis result. To define adaptable services, variability should be identified. But its representations are not much proposed. With the models, variability of services can be easily recognized in business process and interface models.

Second, the decision model can be utilized as the intermediate artifacts for service adaptation as shown in Figure 7.

![Figure 7. Decision Model as an Intermediate Model for Service Adaptation](image)

The decision model is initiated in analysis and filled with attached tasks in design time. Then, the rules, activities, and other constraints are referred in service adaptation when the services are called at run time.

For service adaptation, components for service adaptation such as broker or intermediate components should be defined. And the adaptation component should refer adaptation relevant information.

The modeled variability in this approach can be compatibly used in implementing service adaptation since it conforms to XML representation which is standard representation in SOA.

### 6. Conclusion

The paradigm of Publish-Discover-Compose of SOC presents a challenge on service applicability and adaptability. Since services are not just for predefined clients, rather for potentially many unknown clients, they have to be highly adaptable to various service clients and contexts. To leverage the service reusability and adaptability, the variability presented on services should carefully be modeled by considering the unique computing paradigm and requirements of SOC such as dynamic discovery and composition of services.

In this paper, we presented a framework for modeling service variability by using key constructs of product-line engineering. We presented a technical comparison between the variability on conventional applications and the variability on SOC. Then, we identified four types of variability which may occur on services. As the main contribution of our research, we presented a method to model service variability and design adaptable services for each type of service variability. Using the proposed framework, we believe the applicability and reusability of such services can be greatly increased.

### References


