A Web Services Variability Description Language (WSVL) for Business Users Oriented Service Customization

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. To better facilitate business users in customizing Web services, customization options need to be described at a high level of abstraction. In contrast to related efforts that describe customization options at the technical level of service description, we propose a Web Services Variability description Language (WSVL) that facilitates the representation of such options at business level. The language has several advantages. Firstly, it does not require people, who perform customization, to have knowledge of Web service technologies. Thus, the language enables business users-friendly service customization. Secondly, the language captures not only what can be customized, but also how and where customization operations should happen in a service-oriented way. This self-described property removes the need for a separate procedure for governing service customization. Consequently, this property eases the adoption of the language. We elaborate the design of the language using a case study and describe its usages from both consumers and providers' viewpoints.

Keywords: Service variability, Service customization, Service description language, Feature model, Software Product Line (SPL).

1 Introduction

Services in Service Oriented Architecture (SOA) often contain many variants due to the variability in consumer requirements. In order to address such variability, service customization has proved to be an efficient approach [1-3]. Service customization refers to a *consumer-driven process* of deriving a *service variant*, which contains adequate service capability for a particular consumer, from a *super-service*, which contains a superset of all service capability required for all service variants.

In order to support service customization, customizable services need to be described. However, related work in the literature proposes approaches which are oriented toward IT professionals [2, 3]. In particular, all these approaches are concerned with expressing *variant service capabilities* in the service interface description (i.e. messages, operations, and data types) and exposing those variant service capabilities to service consumers for selection. Customizing services in these

A. Haller et al. (Eds.): WISE 2011 and 2012 Combined Workshops, LNCS 7652, pp. 321–334, 2013.

ways presumes that people who perform service customization have IT background and are very familiar with the technical description of services. And even for IT professionals, these approaches are still challenging because of the large number of variant service capabilities and dependencies among them [1].

In order to facilitate business professionals in customizing services, customization options need to be described at a high-level of abstraction. From business professionals' viewpoint, it is more important to know about what it is that a service variant will achieve, rather than how to technically invoke its capability. In other words, it is more beneficial for business professionals to be able to customize services at the problem space (i.e. business level service variability), rather than the solution space (i.e. technical realization of such variability). To this end, we are developing a feature-based service customization framework that captures and represents service variability at the business level so that it is much easier to customize services [1].

In this paper, we propose a language, namely Web Services Variability description Language (WSVL), for describing customizable services. WSVL adds variability description capability into WSDL, a de facto standard for describing services. Further, a WSVL document is *self-described* and captures not only the information on what customization options are, but also the information on how and where to perform such customization (i.e. the exchange of customization requests and responses) in a service-oriented way. The self-described property removes the need of defining a separate, informal service customization procedure which is likely to vary from one provider to another. Thus, it eases the adoption of the language. In addition to enabling service consumers to customize and consume one service variant, the WSVL language also allows service consumers to manage *variability inter-dependencies* between their applications and customizable partner services in case there are more than two service variants from the same customizable partner service involves.

The rest of the paper is structured as follows. Section 2 presents a case study that will be used throughout the paper to demonstrate the language. We elaborate motivations and designs of different aspects of WSVL in section 3. Section 4 presents the application of WSVL by demonstrating how consumers utilize WSVL documents in customizing services and how providers develop customizable services based on WSVL documents. Section 5 discusses related work and points to future work. We conclude the paper in section 6.

2 Case Study

Swinsure Insurance is an insurance company providing various types of building insurances to various consumers (e.g. insurance brokers, business users or personal users). In exposing its capability as a Web service (namely Swinsure WS), Swinsure Insurance has identified the following variations in its consumer requirements:

- Some consumers only need to get a quote and others will go on to purchase policies.
- Some consumers need to be able to view and update purchased policies.

- There are two policy types, namely residential insurance and business insurance, depending on the purpose of using a building.
- Consumers are able to include extra cover (i.e. accidental damage, fusion cover and extended third party liability) in their policies.
- When purchasing a policy, some consumers choose to use online credit card payment while others prefer to be issued a cover note and pay on invoice.

Note that these variations cannot be arbitrarily combined. Instead, there are dependencies among them. These dependencies come from both consumers' need and Swinsure Insurance's business policies. In particular, Swinsure Insurance has identified the following constraints:

- Those consumers who need to update policies also need to view policies.
- The extended third party liability extra cover is only available to business insurance policies.

In order to efficiently provide this Web service to consumers, Swinsure Insurance decides to develop a customizable service so that consumers can customize the service on their own while satisfying constraints imposed by the provider. In addition, Swinsure Insurance needs to describe its customizable service in a comprehensive and convenient way so that consumers can easily perform the customization. In the following section, we describe how to use WSVL for these purposes.

3 Web Service Variability Description Language (WSVL)

3.1 Overview

A WSVL document contains 4 different sections:

- *Feature Description* section describes the variability of the service by means of features.
- *Customization Description* section describes information related to customization operations.
- *Capability Description* section captures full service capability (i.e. superset of capability of all service variants).
- *Mapping Description* denotes the mapping between variant features and variant service capabilities.

Elements of the *Feature Description* section and the *Mapping Description* section are defined by our WSVL schema. Elements of the *Customization Description* extend the existing elements in WSDL due to semantic similarities. And elements of the *Capability Description* are existing WSDL elements. In the following subsections, we will describe each of these descriptions in detail.

3.2 Feature Description

One essential part of a service variability description is to express what can vary. This involves two types of information [5]. Firstly, what are variant service capabilities?

Secondly, what are dependencies among those variant service capabilities? Dependencies describe mutual inclusion and exclusion relationships among variant service capabilities. From consumers' viewpoint, it is also important that such variability is described at an appropriate level of abstraction for better comprehension. Therefore, in contrast to all related work focusing on capturing variability at the technical level, we support the description of variability at the business level.

To this end, the concepts of features and feature models from the Software Product Line (SPL) research domain well-suit our purpose [6]. SPL is a software engineering paradigm that aims to develop a family of software products from reusable core assets. In SPL, a feature model is used to capture the commonalities and differences among a family of software products [7]. Features are visible characteristics used to differentiate one family member from others. A feature model is a hierarchy 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. Therefore, the use of feature models in capturing business level service variability will provide an appropriate level of abstraction for service customization.

While there are many manifestations of feature modeling techniques, e.g. [8-11], in our work we exploit the cardinality-based feature modeling technique [12, 13]. 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.

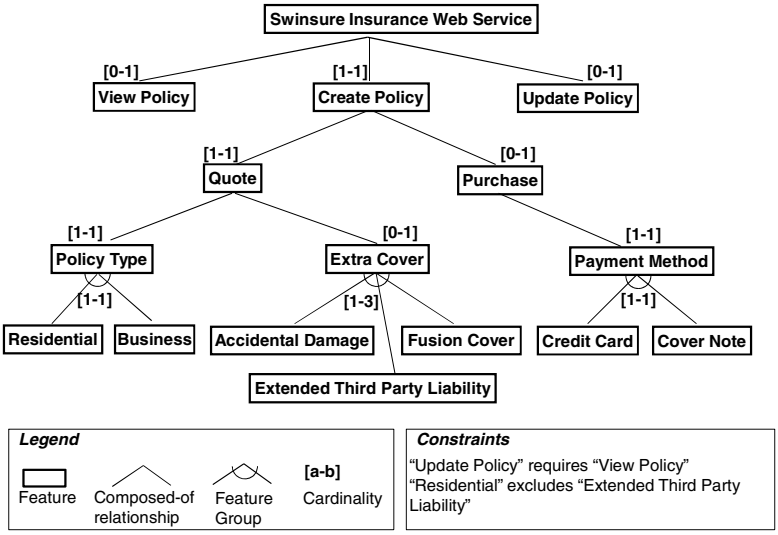

Fig. 1. Feature-based variability representation for the case study

Figure 1 demonstrates a feature model capturing variability of the case study. In the figure, cardinality represented above a feature is feature cardinality, while the one represented below a feature is group cardinality. In this feature model, feature *"Create Policy"* is mandatory (i.e. this feature is required by all consumers) while feature *"View Policy"* and *"Update Policy"* are optional (i.e. the necessity of these features are decided by consumers). Feature *"Create Policy"* is composed by a mandatory feature *"Quote"* and an optional feature *"Purchase"*. *"Policy Type"* is a mandatory feature and is also a feature group with two grouped features, namely *"Residential"* and *"Business"*. The group cardinality of this feature group is [1-1] which implies that consumers have to select one of these two alternative features for their service variants. All other features are described in a similar fashion.

Figure 1 also defines two constraints. These constraints represent dependencies between feature *"Update Policy"* and feature *"View Policy"*, as well as between feature *"Residential"* and feature *"Extended Third Party Liability"*. These constraints and the feature hierarchy precisely capture the variability of Swinsure WS.

The *Feature Description* section in a WSVL document represents the feature model. In particular, it contains description of feature hierarchy and description of feature constraints. Based on the feature description, consumers can specify a *feature configuration* and request a particular service variant. 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.

Figure 2 presents an extracted XML from Swinsure WSVL for a partial feature hierarchy description, the feature constraint description, and a complete feature configuration. The feature configuration describes one consumer with minimum requirements. In this configuration, all optional features are not selected by the consumer when customizing the service.

```
<feature name="CreatePolicy" minCardinality="1"
          maxCardinality="1">
  <feature name="Quote" minCardinality="1"
          maxCardinality="1">
    <feature name="PolicyType" minCardinality="1"
          maxCardinality="1">
      <featureGroup minCardinality="1" maxCardinality="1">
        <feature name="Residential"/>
        <feature name="Business"/>
      </featureGroup>
    </feature>
  </feature>
</feature>
```

Partial feature hierarchy description

```
<featureDescription>
  <featureHierarchy>
    <feature name="SwinsureInsuranceWebService">
      <feature name="CreatePolicy">
        <feature name="Quote">
          <feature name="PolicyType">
            <feature name="Residential"/>
          </feature>
        </feature>
      </feature>
    </feature>
  </featureHierarchy>
</featureDescription>
```

A feature configuration

```
<featureConstraint>
  <constraint>
    <constraintDesc>if (//UpdatePolicy) then (//ViewPolicy) else true();</constraintDesc>
  </constraint>
  <constraint>
    <constraintDesc>if (//Residential) then not (//ExtendedThirdPartyLiability) else true();</constraintDesc>
  </constraint>
</featureConstraint>
```

Feature constraint description

Fig. 2. Extracted XML for feature model and feature configuration

3.3 Customization Description

In addition to the information about what can vary, a WSVL document needs to define how and where such variation can be requested. This is the information about how consumers should construct customization requests, where they should send those requests, and what should be expected as responses from service providers. The *Customization Description* section in a WSVL document is used for this purpose.

The *Customization Description* defines a set of customization operations that consumers may use to customize the service. One typical operation is the one that accepts a feature configuration as the request and returns a reference to WSDL description of a service variant as the response. However, there are many other possible customization operations. For instance, a customization operation might accept an incomplete feature configuration and return a revised WSVL document. This usage enables multi-stage service customization. Or service providers can group resolutions of several variant features into predefined packages. A service consumer can send one package as the request and the provider responses with an updated WSVL. Because of this variety, we have made the *Customization Description* generic, rather than confining it to describing only typical operations.

Customization operations can be described using conventional elements in WSDL due to the semantic similarity between the *Customization Description* and conventional service description. However, in order to clearly separate service operations for service customization and service operations for service consumption, we extend existing elements in WSDL to describe customization operations. Figure 3 presents an extract of Swinsure WSVL for the customization description. The customization operation *"swinsureCustomizationOperation"* is specified using element <<*wsvl:operation*>> which inherit the element <<*wsdl:operation*>>. Similarly, <<*wsvl:portType*>> is used to enclose all customization operations. Further, <<*wsvl:binding*>> specifies the binding of customization operations to transport and messaging protocols which are HTTP and SOAP respectively in the example. Lastly, <<*wsvl:port*>> specifies the endpoint to which a consumer can exchange customization requests and responses. As can be seen from the example, the *Customization Description* section provides sufficient information for governing service customization. Consequently, it makes WSVL documents self-described.

```
<wsvl:portType name="swinsureCustomizationPortType">
  <wsvl:operation name="swinsureCustomizationOperation"/>
</wsvl:portType>
<wsvl:binding name="swinsureBinding" type="swinsureCustomizationPortType">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http" />
  <wsvl:operation name="swinsureCustomizationOperation"/>
</wsvl:binding>
<wsdl:service name="swinsureCustomizationFrontend">
  <wsvl:port binding="swinsureBinding" name="swinsurePort">
    <soap:address location="http://localhost:9000/swinsureCustomizationFrontend" />
  </wsvl:port>
</wsdl:service>
```

Fig. 3. Extracted XML for customization description

3.4 Capability Description

One typical usage of customizable service descriptions is allowing service consumers to customize services. In such situation, the combination of feature description and customization description is sufficient. However, we also consider an advanced use of a customizable service in which the service is used in another customizable service and there are dependencies among their variants. In particular, the derivation of one variant of a customizable composite service requires a particular variant of a customizable partner services. We call such dependency as *variability inter-dependencies* [15]. To support variability inter-dependencies, WSVL needs to incorporate additional information explained in this section and the next one.

Let us consider an example in Figure 4. Swinbroker is building a customizable insurance quoting business process that reuses Swinsure WS. Depending on the type of policy its users request (i.e. residential or business), the Swinbroker will derive different business process variants that interact with different variants of Swinsure WS. In particular, for users requesting residential quoting, the Swinbroker will customize Swinsure WS with feature *"Residential"* enabled and feature *"ExtraCover"* disabled. The derived process variant will invoke the operation *getQuoting4Residential()* of the corresponding service variant. And for users requesting business quoting, the Swinbroker will customize Swinsure WS with feature *"Business"* enabled and feature *"ExtraCover"* disabled to be able to invoke the operation *getQuoting4Business()* of another service variant.

In such situation, it is necessary that Swinbroker knows not only the feature model of Swinsure WS, but also the consequence of customizing the service based on the feature model. That is, how they should interact with service variants which are the result of service customization. And such information should be made available before the service is actually customized (e.g. while Swinbroker models its business process). To this end, the WSVL document needs to describe full capability of a customizable service [16]. This full capability is the superset of capability of all service variants. We reuse WSDL elements for this purpose.

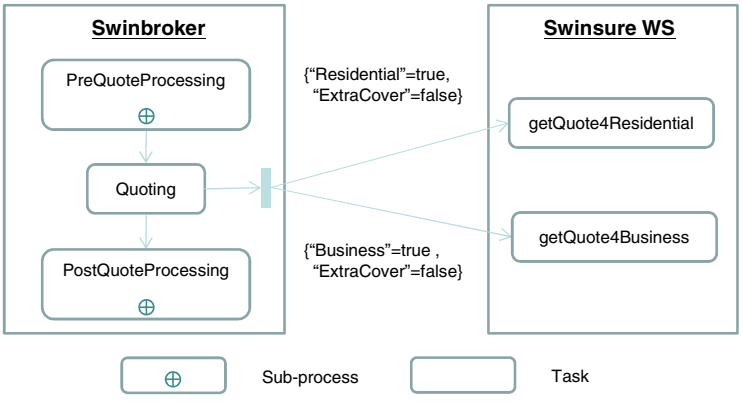

Fig. 4. Variability inter-dependencies between Swinbroker process and Swinsure WS

```
<wsdl:portType name="quotingPortType">
  <wsdl:operation name="getQuote4Residential"/>
  <wsdl:operation name="getQuote4Business"/>
</wsdl:portType>
<wsdl:portType name="purchasingPortType“>
  <wsdl:operation name=“purchasePolicyByCreditCard"/>
  <wsdl:operation name=" purchasePolicyByCoverNote"/>
</wsdl:portType>
```

Fig. 5. Extracted XML for service capability description

Figure 5 presents an excerpt of the capability description for Swinsure WS. There are two <<wsdl:portType>> of which the *“quotingPortType”* is available to all consumers while the *“purchasingPortType”* is only available if feature *“Purchase”* is selected. Among two operations of the *“quotingPortType”*, the operation *“getQuote4Residential”* (or *“getQuote4Business”*) is only available if the corresponding feature *“Residential”* (or *“Business”*) is selected and feature *“ExtraCover”* is disabled. Similarly, for two operations of the *“purchasingPortType”*, the operation *“purchasePolicyByCreditCard”* (or *“purchasePolicyByCoverNote”*) is only available if the corresponding feature *“CreditCard”* (or *“CoverNote”*) is selected.

3.5 Mapping Description

The full service capability description is the superset of the capability of all service variants. However, a WSVL document also needs to explicitly describe what capabilities are available if a feature is selected or disabled. We call a condition over a set of features from which we decide the existence of a variant service capability as *presence condition*. For example in Figure 4, the operation *“getQuote4Residential”* operation only exists in service variants if feature *“Residential”* is true (consequently feature *“Business”* is false because these two features are alternative) and feature *“ExtraCover”* is false. Or for the extracted XML in Figure 5, the portType *“purchasingPortType”* only exists in service variants if feature *“Purchase”* is true. Therefore, a presence condition can be expressed as a conjunction over a set of features involved.

A presence condition needs to be associated with relevant variant service capabilities. To this end, we introduce the concept of links as a mapping between a feature and relevant elements [17]. Each link has an additional attribute specifying whether feature should be selected or disabled for the presence of the elements. Since a presence condition is a conjunction over a set of features, the association of a presence condition and elements can be expressed as a set of links mapping relevant features and the elements. This approach does not require the use of a certain condition expression language for representing the presence condition in a WSVL document. Thus, we can avoid imposing more constraints on consumers in order to interpret the WSVL document.

```
<mappingInfo>
  <link name="LResidential">
    <featureRef ref="fd:Residential"  presence="true"/>
    <serviceElementRef ref="tns:getQuote4Residential" target="operation" />
  </link>
  <link name="LExtraCover">
    <featureRef ref="fd:ExtraCover"  presence="false"/>
    <serviceElementRef ref="tns:getQuote4Residential" target="operation" />
  </link>
  <link name="LPurchase">
    <featureRef ref="fd:Purchase"  presence="true"/>
    <serviceElementRef ref="tns:purchasingPortType" target="portType" />
  </link>
</mappingInfo>
```

Fig. 6. Extracted XML for mapping description

Figure 6 presents an extracted XML from the mapping description section of Swinsure WSVL. The first two links associates feature *"Residential"* and feature *"ExtraCover"* with the operation *"getQuote4Residential"* operation with the *"presence"* attribute of <<*featureRef*>> elements are *"true"* and *"false"* respectively. These two links together specifies above mentioned presence condition that the operation "*getQuote4Residential"* is available only if *"Residential"* is true and *"ExtraCover"* is false. Similarly, the third link specifies that the portType *"purchasingPortType"* only exists if feature *"Purchase"* is true.

With regard to the types of element that will be objects of presence conditions, we consider only port types or operations. While there might be variants in message formats and data types that require presence conditions, we argue that those variabilities should be escalated to variabilities in operations (i.e. a separate operation for each variant message format/data types) and port types. This escalation enables simpler implementation of WSVL-based customizable services because in general each operation is transformed to a method in the service implementation [18]. As shown in Figure 6, types of elements are described using the attribute *"target"* of the corresponding <<*serviceElementRef*>>.

4 Application of WSVL

In order to illustrate the feasibility and applicability of WSVL we describe how service consumers and service providers utilize a WSVL document for service customization. While the mechanisms described in following subsections can be generalized, we just use them as one solution for validating the usefulness and the feasibility of the WSVL language. The explanation is based on Swinsure WSVL.

4.1 WSVL for Service Consumers

While WSVL uses XML which is a machine-readable format, many business professionals are not familiar with writing XML documents manually given an XML schema. Therefore, customizing services described in WSVL will be easier for

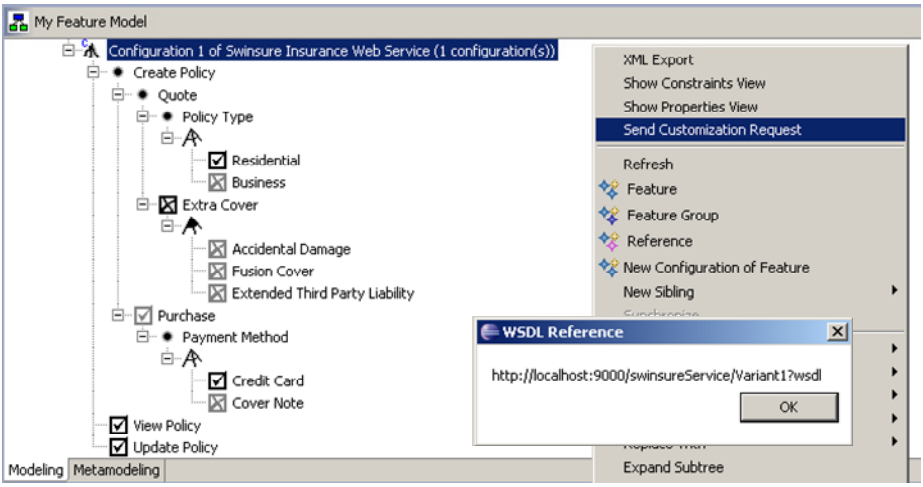

Fig. 7. Customizing services by business professionals

business professionals with the use of appropriate tools. We describe one such tool in this subsection. The tool is compatible with the WSVL schema to demonstrate the language's feasibility.

Figure 7 presents a screenshot of how business professionals can customize Swinsure WS in an intuitive and simple way. We extend the open source feature modeling tool to implement this plugin [19]. Information from the variability description section in the WSVL document is used to reproduce feature model on the consumer side. Feature model is then rendered to the business professionals through an interactive interface so that they can select features they need and remove features they do not need. As there exist feature constraints in the feature model, such constraints are used to validate business professionals' selection, as well as automatically propagate the selection through the feature configuration [14]. This helps to reduce the time and overhead during the customization process, as well as avoiding mistakes. For example, when a business professional selects feature "Residential", feature "Extended Third Party Liability" is automatically disabled due to the constraint between the two features (cf. Figure 1). Once the business professional finishes the customization, he can send a request for a service variant based on the generated feature configuration.

The feature configuration will be embedded in a SOAP message and be sent to the customization endpoint as specified in the customization description section of the WSVL document. As the result, the consumer will be issued an URL from which the consumer can retrieve the WSDL document of the generated service variant.

4.2 WSVL for Service Providers

We have implemented Swinsure WS as an atomic service. Figure 8 presents its software architecture. Swinsure WS has a customization Web service frontend that

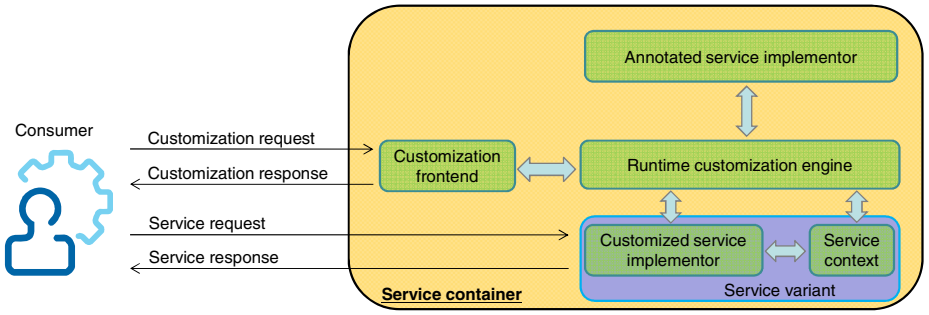

Fig. 8. Architecture for Swinsure WS

accepts a customization request (i.e. a feature configuration) from service consumers. The feature configuration is then passed to the *Runtime customization engine* which dynamically generates and deploys a service variant from an *Annotated service implementor* (described below). The reference to the WSDL of the generated service variant is returned to service consumers through the customization frontend. Consumers can then consume the service by invoking the service variant.

The *Customization frontend* is the Web service implementation of the customization description in the WSVL document. The *Annotated service implementor* is the implementation of service capability description in the WSVL document. The implementation contains additional annotations that specify presence conditions of variant portTypes and variant operations. An example of these annotations is shown with italic underline font in Figure 9. Note that these additional annotations are based on the information on the mapping description section of the WSVL document. The interface *PurchasingPortType* which implements the portType *"PurchasingPortType"* in the WSVL document is only available in a service variant if all features in the *enabledFeatureList* are selected and all features in the *disabledFeatureList* are removed. In this case, the interface is only available if feature *"Purchase"* is true. Similarly, the operation *"PurchasePolicyByCreditCard"* (or *"PurchasePolicyByCoverNote"*) is only available in a service variant if feature *"CreditCard"* (or *"CoverNote"*) is true. Based on these additional annotations, the runtime customization engine can derive and deploy a particular service variant that exposes appropriate service capability. Note that the *Service context* component of a

```
@WebService(name = "purchasingPortType")
@FeatureMapping(enabledFeatureList={"Purchase"}, disabledFeatureList={})
public interface PurchasingPortType {
    @WebMethod
    @FeatureMapping(enabledFeatureList={"CreditCard"}, disabledFeatureList={})
    public PurchasePolicyByCreditCardResponse purchasePolicyByCreditCard(PurchasePolicyByCreditCard request);

    @WebMethod
    @FeatureMapping(enabledFeatureList={"CoverNote"}, disabledFeatureList={})
    public purchasePolicyByCoverNoteResponse purchasePolicyByCoverNote(purchasePolicyByCoverNote request);
}
```

Fig. 9. Annotated service implementor

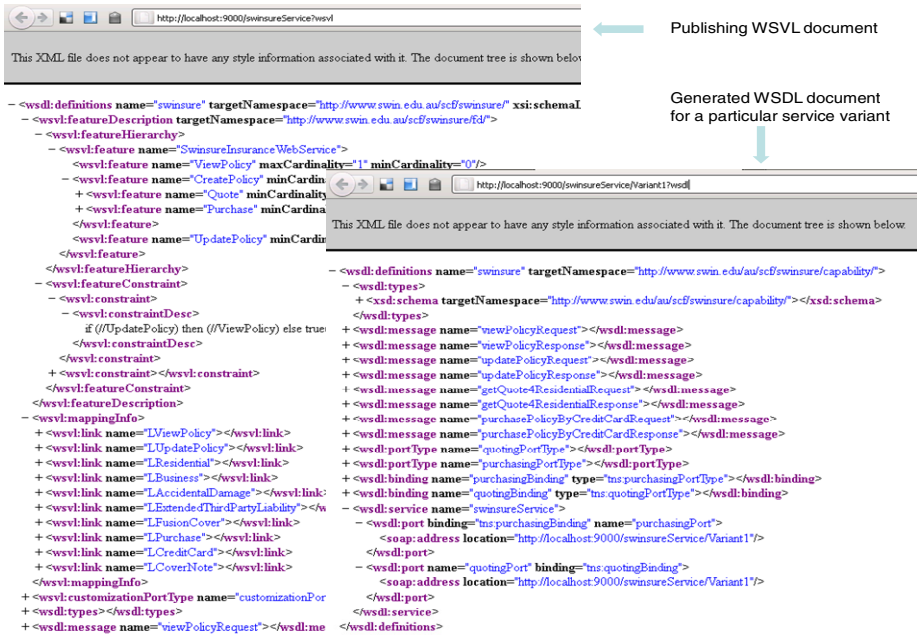

Fig. 10. Snapshots for published WSVL and WSDL documents

service variant stores information about what features are selected and what features are disabled for the service variant. The use of this component enables the correct business logic of the service variant during its execution in case dynamic switching between alternative behaviors for service variants is required.

Figure 10 presents screenshots of how a WSVL document for a customizable service (cf. left panel) and how the generated WSDL document for a particular service variant (cf. right panel) are published. The demonstrated service variant is the response for the feature configuration shown in Figure 7.

5 Discussion and Future Work

Describing customizability of a Web service has been a focus in a number of approaches [2, 3, 20]. Stollberg [2] makes explicit variants of service description elements (e.g. operations or data types) as well as constraints among them so that end users can select the appropriate variant. In contrast, Liang [3] defines customization policies that express which elements in the service description can be changed and what kind of changes is allowed. These customization policies are then used by consumers in customizing the service description. Tosic [20] lists all possible service variants from which consumers can select the most appropriate one. While these approaches are different in the way they represent customization options, they all focus on capturing variability at the technical level of service description. Consequently, the approaches cannot be used by business professionals who have a little knowledge of the underlying Web service technologies.

The work on WSVL is strongly influenced by the ideas of feature models from the SPL research domain [7, 12, 21]. And there are many variability description languages for representing feature models, e.g. [22-25]. Since our service customization framework utilizes the cardinality-based feature modeling technique [1, 12], we have develop the feature description part in the WSVL schema based on concepts of this technique. However, our contribution in developing WSVL is not about the representation of feature models in the WSVL Schema. Instead, we design the language so that it is self-described, comprehensive, and business users-friendly.

We have presented in the paper one way of developing a customizable service based on WSVL documents. We believe that such approach can be generalized to have a semi-automated process for developing customizable services using Model Driven Engineering (MDE) techniques. For example, additional annotations can be added as extension to JAX-WS [18] so that the skeleton of the service implementation (i.e. annotated service implementor) can be automatically generated from a WSVL document. Further, the software architecture for customizable services can be revised to have a middleware supporting customizable services.

6 Conclusion

In this paper, we define a Web Services Variability description Language (WSVL) that can be used to describe customizable services. The language facilitates business professionals to customize services on the consumer side by raising the level of abstraction at which customization options are described. To this end, we exploit the concept of feature models from SPL so that business professionals can reason about and customize services at the business level. The language is self-described because it describes not only what can be customized (i.e. feature description section), but also how and where such customization operations can be performed (i.e. customization description section). Furthermore, the language can be used to produce and consume one particular service variant, as well as support variability inter-dependencies between the service and other customizable service when more than one variant is involved (i.e. mapping description and capability description sections). The usage of the language is demonstrated thoroughly using a case study. We also describe how the language can be used by both consumers and providers with respect to service customization. As the future work, we plan to derive techniques for facilitating service providers in developing and deploying customizable services based on WSVL documents.

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. Nguyen, T., et al.: A Feature-Oriented Approach for Web Service Customization. In: IEEE International Conference on Web Services, pp. 393–400 (2010)
2. 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)

3. 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)
4. Christensen, E., et al.: Web Services Description Language (WSDL) 1.1, March 15 (2001), http://www.w3.org/TR/wsdl
5. Schmid, K., et al.: A customizable approach to full lifecycle variability management. Science of Computer Programming 53(3), 259–284 (2004)
6. Pohl, K., et al.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer-Verlag New York, Inc. (2005)
7. Kang, K.C., et al.: Feature-Oriented Domain Analysis (FODA) Feasibility Study, in Technical Report, Softw. Eng. Inst., CMU. p. 161 pages (November 1990)
8. 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)
9. Griss, M.L., et al.: Integrating Feature Modeling with the RSEB. In: Proceedings of the 5th International Conference on Software Reuse. IEEE Computer Society (1998)
10. Schobbens, P.-Y., et al.: Generic semantics of feature diagrams. Comput. Netw. 51(2), 456–479 (2007)
11. Kang, K.C., et al.: FORM: A feature-oriented reuse method with domain-specific reference architectures. Ann. Softw. Eng. 5, 143–168 (1998)
12. Czarnecki, K., et al.: Formalizing cardinality-based feature models and their specialization. Software Process: Improvement and Practice 10(1), 7–29 (2005)
13. Czarnecki, K., et al.: Cardinality-based feature modeling and constraints - a progress report. In: Proceedings of International Workshop on Software Factories, OOPSLA (2005)
14. Benavides, D., et al.: Automated analysis of feature models 20 years later: A literature review. Information Systems 35(6), 615–636 (2010)
15. 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)
16. Czarnecki, K., Antkiewicz, M.: Mapping Features to Models: A Template Approach Based on Superimposed Variants. In: Glück, R., Lowry, M. (eds.) GPCE 2005. LNCS, vol. 3676, pp. 422–437. Springer, Heidelberg (2005)
17. Didonet, M., et al.: Weaving Models with the Eclipse AMW plugin. In: Proceedings of Eclipse Modeling Symposium, Eclipse Summit Europe (2006)
18. Kotamraju, J.: JSR224 - The Java API for XML-Based Web Services (JAX-WS) 2.2, December 10 (2009), http://jcp.org/aboutJava/communityprocess/mrel/jsr224/index3.html
19. Czarnecki, K.: Feature Modeling Plug-in, http://gsd.uwaterloo.ca/projects/fmp-plugin/
20. Tosic, V., et al.: WSOL — Web Service Offerings Language. In: Web Services, E-Business, and the Semantic Web, pp. 57–67 (2002)
21. Chen, L., et al.: Variability management in software product lines: a systematic review. In: Proc. of the 13th Int. Software Product Line Conf. (2009)
22. Boucher, Q., et al.: Introducing TVL, a Text-based Feature Modelling Language. In: Proc. of the 4th Int. Workshop on Variability Modeling of Soft. Intensive Syst., pp. 159–162 (2010)
23. Mendonca, M., et al.: S.P.L.O.T.: software product lines online tools. In: Proceeding of the 24th ACM SIGPLAN Conference Companion on OOPSLA. ACM (2009)
24. Benavides, D., et al.: On the Modularization of Feature Models. In: First European Workshop on Model Transformation (2005)
25. Cechticky, V., et al.: XML-Based Feature Modelling, in Software Reuse: Methods, Techniques and Tools, pp. 101–114 (2004)