An Approach to Domain-Specific Reuse in Service-Oriented Environments

Jianwu Wang[1,2], Jian Yu[1], Paolo Falcarin[1], Yanbo Han[3], and Maurizio Morisio[1]

[1] Software Engineering Research Group, Dept. of Control and Computer Engineering, Politecnico di Torino, 10129, Torino, Italy
[2] San Diego Supercomputer Center, University of California, San Diego, 92093, USA
[3] Research Centre for Grid and Service Computing, Institute of Computing Technology, Chinese Academy of Sciences, 100080, Beijing, China
wangjianwu@gmail.com,
{jian.yu,paolo.falcarin,maurizio.morisio}@polito.it,
yhan@ict.ac.cn

Abstract. Domain engineering is successful in promoting reuse. An approach to domain-specific reuse in service-oriented environments is proposed to facilitate service requesters to reuse Web services. In the approach, we present a conceptual model of domain-specific services (called *domain service*). Domain services in a certain business domain are modeled by semantic and feature modeling techniques, and bound to Web services with diverse capabilities through a variability-supported matching mechanism. By reusing pre-modeled domain services, service requesters can describe their requests easily through a service customization mechanism. Web service selection based on customized results can also be optimized by reusing the pre-matching results between domain services and Web services. Feasibility of the whole approach is demonstrated on an example.

Keywords: Domain-Specific Reuse, Domain Service Model, Service Capability Diversity, Variability-Supported Service Matching, Service Customization.

1 Introduction

Service orientation is becoming a dominant paradigm in distributed computing. There are a large amount of available Web services on the Internet, and there will be more. In the bioinformatics domain, for instance, the number of Web services has added up to over 3000 [1]. On the one hand, the abundance of Web services facilitates on-demand application construction; on the other hand, since Web services are implemented and maintained independently, slight differences among them bring difficulties for their (re)use.

We will use a simplified example from the weather service domain throughout the paper (see Fig. 1). There are over 15 real Web services (including 179 independent operations)[1] providing weather forecast on the Internet. When the number of Web

[1] An incomplete list can be found at http://wangjianwu.googlepages.com/webservicelistforweatherforecast

services with similar functionality is huge, it is very difficult for service requesters to directly select proper services and reuse them.

We can then split the problem into two parts:

1) *Similarity and diversity of service requests*[2]: As shown in Fig.1, *Req1* and *Req2* are two similar yet different service requests. For instance, wind speed information is mandatory in *Req1* but not in *Req2*; the target location areas and the preferred ways to describe locations are also different. A key problem at the service request level is how to facilitate service requesters to describe their service requests in a certain business domain where service requests are similar yet diverse.
2) *Similarity and diversity of Web services*: Also as shown in Fig.1, *WS1* and *WS2* are two similar yet different Web services. For instance, *WS1* can only forecast weather in the U.S., while *WS2* can forecast weather worldwide; their input parameters for location are also different; moreover, *WS2* has an additional output: wind speed. A key problem at the Web service level is how to optimize on demand selection of executable Web services in a service-oriented environment where Web services are abundant yet diverse in capability (namely Input, Output and QoS).

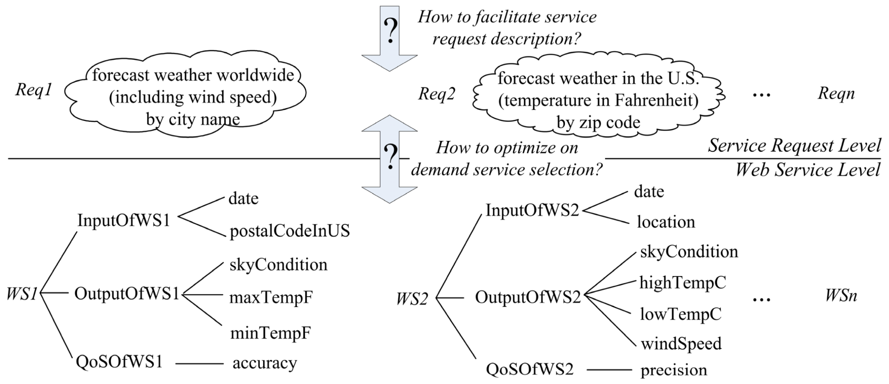

Fig. 1. Two levels of service usage in service-oriented environments

To tackle the above problems, we propose an approach to domain-specific reuse in service-oriented environments based on our previous work [2, 3]. The core of this approach is a conceptual model of domain-specific services (called *domain service*), which acts as a broker between service requesters and Web services. The following advantages can be obtained:

1) *Simplifying service request description by reusing pre-modeled domain services*: Feature modeling techniques [4, 5] are used in domain services to model the commonalities and variabilities of similar service capabilities. So instead of describing service requests from scratch, particular service requests can easily be described by reusing pre-modeled domain services.

[2] Service requests in this paper are restricted to single services. Complex service requests can be met through service composition, which is beyond the scope of this paper.

2) *Accelerating service request satisfaction by reusing pre-matching results between domain services and Web services*: With pre-modeled domain services, Web services can also be matched to proper domain services in advance. Then the matching between particular service requests and Web services can be optimized by reusing the pre-matching results between domain services and Web services.

The rest of this paper is organized as follows. Firstly, we discuss related work in Section 2. Then an overview of the approach is given in Section 3. Two parts of our approach, namely domain engineering process and application engineering process, are explored in detail in the following two sections. Finally, we conclude the paper in Section 6.

2 Related Work

Our approach can be seen as a kind of domain modeling applied to service-oriented environments in order to facilitate service request description and service matching.

Recently, some traditional approaches in requirement engineering researches have been applied on service request modeling, such as goal oriented [6] and value based [7]. Yet they do not tackle how to reuse service requests. There are also some researches addressing the importance of combining top-down requirement refinement and bottom-up existing service resource reuse [8, 9]. Our work also follows this way, and our work emphasizes the variability modeling of similar services which is omitted in the above work.

Feature modeling in domain engineering approaches has been proved to be successful in representing reusable and configurable requirements for its good capacity to express commonalities and variabilities [4, 5]. Recently, some effort has been put into importing feature modeling to model some aspects of commonalities and variabilities in service-oriented environments. In [10], each feature represents a service operation, which can support operation variabilities in similar systems. Feature modeling is also used to express non-functional properties [11, 12] and implementation techniques [13] of services. But none of the above proposals deal with service capability variability, which is a main difficulty for service requesters to directly select Web services.

There has been much research on service matching [14, 15, 16], however they usually assume that there is a given service request and an available service set, and emphasize on matching degree and theory foundation. How to reuse pre-matching results to facilitate future service matching is still an open challenge.

There are also some works on service virtualization [17, 18], which focus on how to abstract similar services for better (re)use. However, the abstraction mechanisms are rather rigid. For example, *WS1* and *WS2* in Fig. 1 can not be abstracted into one abstract weather forecast service for the capability differences between them. In a service-oriented environment where there are abundant Web services with diverse capabilities, these mechanisms can only bring limited promotion in reuse.

3 Overview of the Approach

Referring to the software development process in traditional domain engineering approaches [5], our approach also consists of a domain engineering process and an application engineering process (shown in Fig. 2). Activities (rectangles in Fig. 2) and deliverables (italics in Fig.2) in this approach will be outlined in this section and explained in detail in the following two sections.

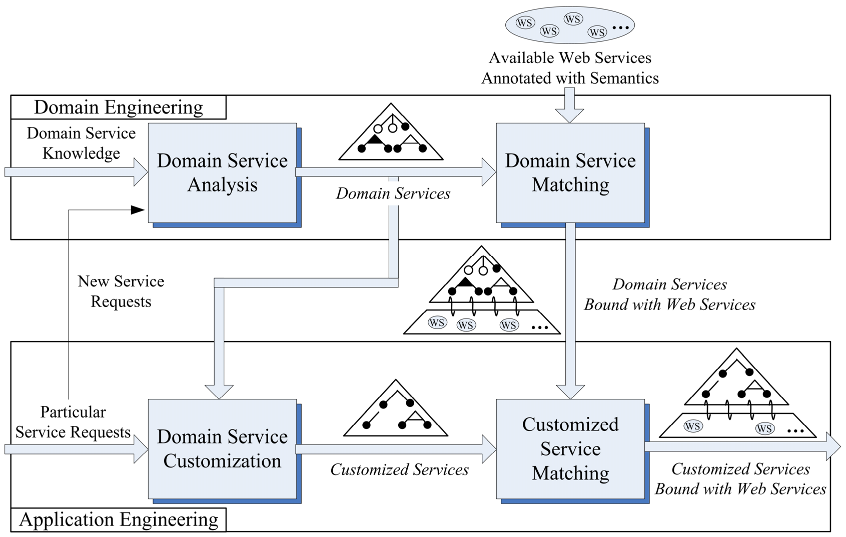

Fig. 2. Overview of the approach to domain-specific reuse in service-oriented environments

Domain Engineering Process: This process is to define domain services and bind them with Web services for future reuse. Firstly, domain services are modeled by domain experts through domain service analysis. Secondly, Web services are bound to proper domain services through service capability matching.

Application Engineering Process: This process is to reuse the deliverables generated in the domain engineering process in order to improve the satisfaction of particular service requests. Particular service requests are firstly described by customizing proper domain services, which can be made easier through reusing pre-modeled domain services. Secondly, suitable Web services are bound to customized services by customized service matching, which can be optimized through reusing pre-matching results between domain services and Web services. Then each Web service bound to customized services can be executed to perform the corresponding service requests.

For the applicable domains, our practice shows that the approach is suitable for the business (sub)domains, such as bioinformatics and travel information domain, which have the following characteristics: 1) Service requesters want to describe their personal-ized requests; 2) It is easy to define domain scope, and model domain on-tologies and services; 3) There are a large amount of available Web services provided by different organizations.

4 Domain Engineering Process

To discuss our domain engineering process, this section is divided into two subsections by its main activities.

4.1 Domain Service Analysis

Referring to traditional domain analysis activities, the domain service analysis also involves two main sub activities: domain service identification and domain service modeling. The first sub activity can refer to existing approaches, such as [8], and is omitted here. In the second sub activity, capability information of identified domain services is modeled for the matching with that of Web services. Here, the commonalities and variabilities of domain service capabilities are modeled by features to facilitate future service request description, and parameter semantics of domain services are annotated by domain ontology concepts for automatic and exact service matching. Besides, since domain ontologies express shared concepts in the domain, it is easy for service requesters to understand domain services.

To discuss our domain services in more detail, related formal definitions are given below, and the corresponding schemas in XML can easily be obtained from the definitions:

Def. 1 (Feature): *feature* = < *FeatureNode*, *FeatureEdge* >, *FeatureNode* = {*supernode*} ∪ *SubFeatureNode*, *SubFeatureNode* ∈ {$subnode_1$, ... , $subnode_n$}, *FeatureEdge* = { <*sn*, *sfn*, *ft*> | *sn=supernode*, *sfn* ∈ *SubFeatureNode*, *ft* ∈ {*Man*, *Opt*, *XOR*, *OR*}}. $subnode_i$ and the corresponding feature edges start from $subnode_i$ also form a feature (called *sub feature*). Then a feature with all its descendent features forms a feature tree.

Def. 2 (Domain Service): *domainservice* = < *inputFeature*, *outputFeature*, *qosFeature* >. Hereinto, *inputFeature*, *outputFeature* and *qosFeature* are all features. And all the elements of *SubFeatureNode* of *inputFeature/outputFeature/qosFeature* are annotated with proper domain ontology concepts.

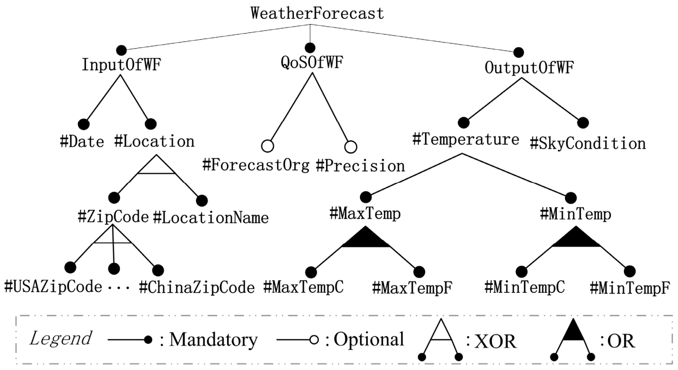

Fig. 3. A domain service example with capability variability

For the example of weather forecast, a simplified domain service in weather service domain, *WeatherForecast*, is modeled (shown in Fig. 3). For instance, typical location of *WeatherForecast* is expressed as *ZipCode* or *LocationName*, but not both. They are thus modeled as two sub features of *Location*, and the feature type is XOR. The partial formal definition of *WeatherForecast* is as follows:

WeatherForecast = <*inputFeatureOfWF*, *outputFeatureOfWF*, *qosFeatureOfWF*>
inputFeatureOfWF = < { *InputOfWF*, *Date*, *Location* }, { <*InputOfWF*, *Date*, *Man*>, <*InputOfWF*, *Location*, *Man* > } >
locationFeature = < { *Location*, *ZipCode*, *LocationName* }, { <*Location*, *ZipCode*, *XOR*>, <*Location*, *LocationName*, *XOR*> } >
...

4.2 Domain Service Matching

Instead of separate domain implementations according to domain models in traditional domain engineering approaches, we think it is better to keep an eye on available Web services as well, which is also addressed in [8, 9]. So we employ a service matching mechanism to carry out domain implementation in service-oriented environments, which matches and binds domain services with proper executable Web services for future reuse. This way also realizes the seamless integration between the outputs from domain analysis and the inputs needed for domain implementation.

To enable automatic and exact service matching with domain services, techniques of semantic Web services [14, 19] are used. Parameters of Web services are all annotated with domain ontology concepts. Our definition on semantic Web service is given below and Fig.4 shows the corresponding semantic Web services of *WS1* and *WS2*. Note that, to be more precise, it should be semantic Web service operation. We use semantic Web service just for short.

Def. 3 (Semantic Web Service): *sws* = <*invokeUrl*, *InputPara*, *OutputPara*, *QoSPara*>. Hereinto, *invokeUrl* is the URL for service invocation; *InputPara*/*OutputPara*/*QoSPara* is the set of Input/Output/QoS parameters which are all annotated with proper domain ontology concepts.

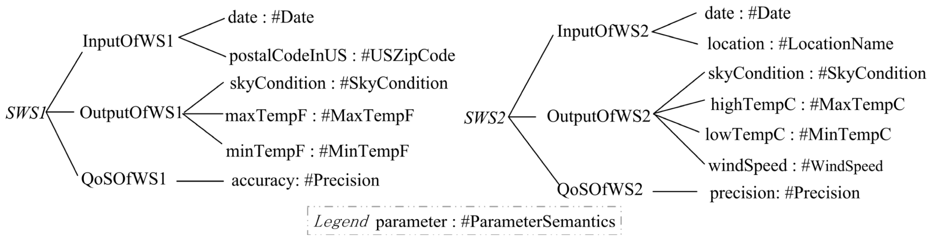

Fig. 4. Examples of semantic Web services

To adapt to the capability diversity of similar Web services, we employ a variability-supported service matching mechanism. Each domain service is shown as a feature tree. And a feature tree can be seen as a kind of AND/OR tree [20] extended with optional and XOR nodes. Then the solvability policy of feature trees can be obtained by extending that of AND/OR trees. Hence the principle of our matching is to firstly semantically match the parameters of Web services with those of domain services, and then to estimate the solvability of domain service feature trees. If a domain service feature tree is solvable on the condition of a certain Web service's capability, it means that the Web service's capability belongs to the capability variability (namely possible capability set) of the domain service, then we say the Web service matches the domain service.

The following formal definitions will be firstly given for detailed discussion.

Def. 4 (Concept Matching): Suppose *concept1* and *concept2* are two ontology concepts. If *concept1* is equal to or subclass of *concept2*, then *concept1* matches *concept2*, which is written as *cm(concept1, concept2) = TRUE.*

Def. 5 (Concept Set Matching): Suppose *Concept1* and *Concept2* are two ontology concept sets. If an injective function exists: {<*x*, *y*>| *x*∈*Concept1*, *y*∈*Concept2*, *cm(x, y) = TRUE*}, then *Concept1* matches *Concept2*, which is written as *csm(Concept1, Concept2) = TRUE.*

Def. 6 (Feature Tree Solvability Policy): If the feature type between one feature *feature* and its sub features is mandatory or optional, then *feature* is solvable if and only if all its mandatory sub features are solvable; if the feature type is OR, then *feature* is solvable if and only if one or more of its sub features are solvable; if the feature type is XOR, then *feature* is solvable if and only if one of its sub features is solvable. A feature tree is solvable if and only if its root feature is solvable.

Def. 7 (Feature Solvability): Suppose *feature* is a feature and *Feature* is a feature set. Given all the elements of *Feature* are solvable, if *feature* is solvable according to Def. 6, then *feature* is solvable on the condition of *Feature*, which is written as *fs(Feature, feature) = TRUE.*

Def. 8 (Semantic Solvability of Feature): Suppose *feature* is a feature, *Concept* is a concept set. If there exists a feature set *Feature* (its annotated concept set is written as *FeatureConcept*) such that *(csm(Concept, FeatureConcept)* $\wedge$ *fs(Feature, feature)) = TRUE*, then *feature* is semantically solvable on the condition of *Concept*, which is written as *ss(Concept, feature) = TRUE.*

From the above definitions, we can get the following function to estimate the solvability of a feature *y* on the condition of a concept set *x*. It is a recursive function that the solvability of a feature depends on its semantic matching or the solvability of its sub features.

$$ss(x,y)=\begin{cases} m(x,y) & y\in LF \\ m(x,y)\vee(ss(x,sub_{11}(y))\wedge ss(x,sub_{12}(y))\wedge\ldots\wedge ss(x,sub_{1n}(y))) & y\notin LF\wedge(ft(y,Sub_1(y))=Man) \\ & \wedge(ft(y,Sub(y)-Sub_1(y))=Opt) \\ m(x,y)\vee(ss(x,sub_1(y))\wedge\neg ss(x,sub_2(y))\wedge\ldots\wedge\neg ss(x,sub_n(y))) & \\ \quad\vee(\neg ss(x,sub_1(y))\wedge ss(x,sub_2(y))\wedge\ldots\wedge\neg ss(x,sub_n(y)))\vee & \\ \quad\ldots\vee(\neg ss(x,sub_1(y))\wedge\neg ss(x,sub_2(y))\wedge\ldots\wedge ss(x,sub_n(y))) & y\notin LF\wedge(ft(y,Sub(y))=XOR) \\ m(x,y)\vee ss(x,sub_1(y))\vee ss(x,sub_2(y))\vee\ldots\vee ss(x,sub_n(y)) & y\notin LF\wedge(ft(y,Sub(y))=OR) \end{cases}$$

Hereinto, *LF* is the leaf feature set whose elements do not have sub features; *Sub(y)* is the sub feature set of *y* whose elements are sub_1, …, sub_n; $Sub_1(y)$ is a sub set of *Sub(y)* whose elements are sub_{11}, …, sub_{1n}; *ft(y, Sub(y))* is the feature type between feature *y* and its sub features.

Def. 9 (Service Matching): Suppose *sws* is a semantic Web service, and *ds* is a domain service. The annotated concept set of *sws*'s Input/Output/QoS parameters is written as *sws.InputConcept/OutputConcept/QoSConcept.* If *(ss(sws.InputConcept, ds.inputFeature)* $\wedge$ *ss(sws.OutputConcept, ds.outputFeature)* $\wedge$ *ss(sws.QoSConcept, ds.qosFeature)) = TRUE*, then *sws* matches *ds*, which is written as *sm(sws, ds) = TRUE.*

The concrete service matching algorithm can easily be obtained from the above definitions and is then omitted for the space limitation.

Besides semantic and variability-supported, another property of the matching mechanism can also be obtained from the definitions, called *Additional Parameter Allowed.* Based on the above definitions, if *csm(Concept1, Concept2) = TRUE*, and *Concept1* $\subseteq$ *Concept1', then csm(Concept1', Concept2) = TRUE.* So domain services can match Web services with additional parameters. This property fits the characteristic that independent Web services may have additional parameters compared to pre-modeled domain services.

For the above weather forecast example, both *SWS1* and *SWS2* matches domain service *WeatherForecast.* Let's take the input matching between *WeatherForecast* and *SWS1* for instance (Fig. 5), the annotated concept set of input parameters of *SWS1* matches feature set:{#Date, #USZipCode} (based on Def. 4), and the input feature of *WeatherForecast* is semantically solvable on the condition of {#Date, #USZipCode} according to Def. 8. So *ss(SWS1.InputConcept, WeatherForecast.inputFeatureOfWF) = TRUE.*

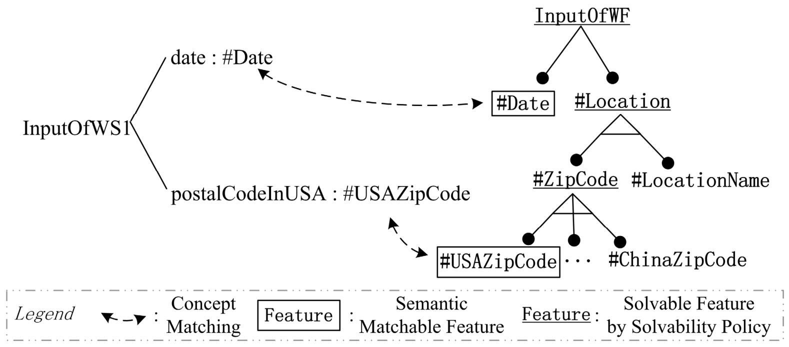

Fig. 5. Example of service matching between domain services and semantic Web services

5 Application Engineering Process

To reuse the deliverables generated in the above domain engineering process to facilitate the satisfaction of particular service requests, a corresponding application engineering process is discussed in this section. We will discuss it through two subsections according to its main activities.

5.1 Domain Service Customization

With reusable domain services, service requesters need not describe their requests from scratch. Yet there still may be a few differences between particular service requests and domain services. So we employ a domain service customization mechanism to enable service requesters to describe their requests by reusing domain services.

Based on existing works on feature configuration [21], service customization operations are defined, which can be classified into three categories: Add (e.g. add one mandatory sub feature), Delete (e.g. delete one optional sub feature) and Configure (e.g. select one sub feature from a XOR feature). All operations can be listed and formally defined following the way of the example below.

$$Feature \times SubFeatureNode \xrightarrow{selectXORFeature} Feature : \{ <x, y, z> \mid x \in Feature,\ y \in x.SubFeatureNode,\ z \in Feature,\ z.FeatureNode = \{ y \} \cup \{ x.supernode \},\ z.FeatureEdge = \{ <x.supernode, y, Man> \} \}$$

For *Req1* in the example of weather forecast, the service requester can customize *WeatherForecast* by adding wind speed as a sub feature of its output and selecting the sub feature LocationName of the Location feature. The customized result is shown in Fig.6. For the features she does not customize (such as Centigrade or Fahrenheit), it means they do not concern her, so each possibility of their variabilities is suitable to her.

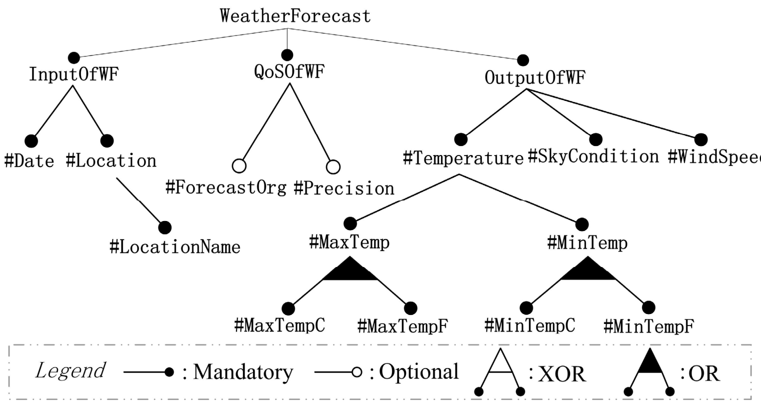

Fig. 6. An example of customized WeatherForecast domain service

5.2 Customized Service Matching

To perform particular service requests, not like product configuration on a separately implemented software in traditional domain engineering approaches, a mechanism to match and select Web services according to the customized service is employed, which can reuse the available Web services.

The service matching algorithm in Section 4 can also be applied in the customized service matching, and we find that pre-matching results of domain services is reusable for some customization operations which can then optimize the service selection.

Theorem 1: Suppose *ds* is a domain service, *op* is a customization operation on *ds*, and *SWS* is a semantic Web service set. The customization result of *ds* by *op* is written as *op(ds)*. For $SWS_1 = \{x \mid x \in SWS,\ sm(x, ds) = TRUE\}$ and $SWS_2 = \{x \mid x \in SWS,$ $sm(x, op(ds)) = TRUE\}$, if the following proposition is true, then $SWS_2 \subseteq SWS_1$:

$$
\begin{aligned}
&(\forall x)(fs(x, op(ds).inputFeature)) \Rightarrow (fs(x, ds.inputFeature)) \\
&\wedge (\forall x)(fs(x, op(ds).outputFeature)) \Rightarrow (fs(x, ds.outputFeature)) \\
&\wedge (\forall x)(fs(x, op(ds).qosFeature)) \Rightarrow (fs(x, ds.qosFeature))
\end{aligned}
$$

Proof: For each element of SWS_2: *sws*, *ss(sws.InputConcept*, *op(ds).inputFeature)*= *TRUE*, which is based on Def. 9. Then, there exists a feature set (written as *Feature*) and a corresponding annotated concept set (written as *FeatureConcept*), such that *(csm(sws.InputConcept*, *FeatureConcept)* $\wedge$ *fs(Feature*, *op(ds).inputFeature))* = *TRUE*. If the above proposition is true, then the following proposition is also true: $(fs(Feature, op(ds).inputFeature)) \Rightarrow (fs(Feature, ds.inputFeature))$. So *(csm(sws.InputConcept*, *FeatureConcept)* $\wedge$ *fs(Feature*, *ds.inputFeature))* = *TRUE*. Then *ss(sws.InputConcept*, *ds.inputFeature)* = *TRUE*, which is according to Def. 8. In similar ways, we can know that *ss(sws.OutputConcept*, *ds.outputFeature)* = *TRUE* and *ss(sws.QoSConcept*, *ds.qosFeature)* = *TRUE*. So *sm(sws*, *ds)* = *TRUE*, namely $sws \in SWS_1$.

For each customization operation, we can formally know whether it makes the proposition of Theorem 1 true. So, if a domain service is customized by the operations making the proposition true, the service selection for the customized service can be optimized. Not all the available Web services, but only Web services bound to the corresponding domain service need to be tested whether they match the customized service. Moreover, Web services can be matched automatically and executed instantly, then particular service requests can be performed on-the-fly.

For *Req1* in the example of weather forecast, both of the needed customization operations (namely *addNewFeature* and *selectXORFeature*) meet the proposition of Theorem 1, so only the Web services bound to *WeatherForecast* need to be tested again using the matching algorithm in sub section 4.2. Of *SWS1* and *SWS2*, only *SWS2* matches the customization result. So it can be executed to perform *Req1*.

6 Conclusions

To promote service reuse from a domain oriented perspective, an approach to domain-specific reuse in service-oriented environments is proposed. Hereinto, domain services in a certain business domain are modeled and matched to proper Web services for reuse in the domain engineering process. Then, new service requests in the same domain can be easily satisfied by reusing pre-modeled domain services and pre-matching results in the application engineering process. Feasibility of the whole approach has been primarily validated through running some sample services in a browser/server architecture-based prototype.

For future work, since the diversity of real service requests and Web services is very complicated, our approach needs to be extended to have more expressive power. We are supporting more complex feature models, such as feature constraints, and more complex service capability description, such as service precondition and effect. Moreover, a more robust and friendly tool, and more and in-depth empirical experiments will be implemented to obtain evidence, which can testify the advantages of our approach.

Acknowledgement

This research is partially funded by the National Natural Science Foundation of China (NSFC) under grand No. 60573117.

References

1. Hull, D., Zolin, E., et al.: Deciding Semantic Matching of Stateless Services. In: 21st National Conference on Artificial Intelligence and 18th Innovative Applications of Artificial Intelligence Conference (AAAI 2006), pp. 1319–1324 (2006)
2. Han, Y., Geng, H., et al.: VINCA - A Visual and Personalized Business-level Composition Language for Chaining Web-based Services. In: Orlowska, M.E., Weerawarana, S., Papazoglou, M.P., Yang, J. (eds.) ICSOC 2003. LNCS, vol. 2910, pp. 165–177. Springer, Heidelberg (2003)
3. Wang, J., Yu, J., et al.: A Service Modeling Approach with Business-Level Reusability and Extensibility. In: 1st IEEE Int. Workshop on Service-Oriented System Engineering, pp. 23–28 (2005)
4. Kang, K.C., Cohen, S.G., et al.: Feature-Oriented Domain Analysis Feasibility Study. Technical Report: SEI-90-TR-21. Pittsburgh, Software Engineering Institute, Carnegie Mellon University (1990)
5. Czarnecki, K., Eisenecker, U.W.: Generative Programming: Methods, Tools and Applications. Addison-Wesley, New York (2000)
6. Lo, A., Yu, E.: From Business Models to Service-Oriented Design: A Reference Catalog Approach. In: Parent, C., Schewe, K.-D., Storey, V.C., Thalheim, B. (eds.) ER 2007. LNCS, vol. 4801, pp. 87–101. Springer, Heidelberg (2007)
7. Gordijn, J., Yu, E., et al.: E-service design using i* and e3 value modeling. IEEE Software 23(3), 26–33 (2006)
8. Arsanjani, A.: Service-Oriented Modeling and Architecture (2004), `http://www.ibm.com/developerworks/library/ws-soa-design1/`
9. Maiden, N.: Servicing Your Requirements. IEEE Software 23(5), 14–16 (2006)
10. Chen, F., Li, S., et al.: Feature Analysis for Service-Oriented Reengineering. In: 12th Asia-Pacific Software Engineering Conference (APSEC 2005), pp. 201–208 (2005)
11. Wada, H., Suzuki, J., et al.: A Feature Modeling Support for Non-Functional Constraints in Service Oriented Architecture. In: 2007 IEEE Int. Conf. on Services Computing (SCC 2007), pp. 187–195 (2007)
12. Fantinato, M., Gimenes, I., et al.: Supporting QoS Negotiation with Feature Modeling. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 429–434. Springer, Heidelberg (2007)
13. Robak, S., Franczyk, B.: Modeling Web Services Variability with Feature Diagrams. In: Chaudhri, A.B., Jeckle, M., Rahm, E., Unland, R. (eds.) NODe-WS 2002. LNCS, vol. 2593, pp. 120–128. Springer, Heidelberg (2003)
14. Martin, D., Paolucci, M., et al.: Bringing Semantics to Web Services: The OWL-S Approach. In: Cardoso, J., Sheth, A.P. (eds.) SWSWPC 2004. LNCS, vol. 3387, pp. 26–42. Springer, Heidelberg (2005)
15. Li., L., Horrocks, I.: A Software Framework for Matchmaking Based on Semantic Web Technology. In: 12th Int. World Wide Web Conference (WWW 2003), pp. 331–339 (2003)

16. Paolucci, M., Kawamura, T., et al.: Semantic Matching of Web Services Capabilities. In: Horrocks, I., Hendler, J. (eds.) ISWC 2002. LNCS, vol. 2342, pp. 333–347. Springer, Heidelberg (2002)
17. Tan, Y., Vellanki, V., et al.: Service Domains. IBM Systems Journal 43(4), 734–755 (2004)
18. Benatallah, B., Sheng, Q., et al.: The Self-Serv Environment for Web Services Composition. IEEE Internet Computing 7(1), 40–48 (2003)
19. McIlraith, S., Son, T., et al.: Semantic Web Services. IEEE Intelligent Systems 16(2), 46–53 (2001)
20. Luger, G.: Artificial Intelligence: Structures and Strategies for Complex Problem Solving, 5th edn. Pearson Addison Wesley, London (2004)
21. Czarnecki, K., Helsen, S., et al.: Staged Configuration through Specialization and Multi-Level Configuration of Feature Models. Software Process: Improvement and Practice 10(2), 143–169 (2005)