Two-Level Service-Oriented Architecture Based on Product-Line

Joonseok PARK†††, Mikyeong MOON††, Nonmembers, and Keunhyuk YEOM††, Member

SUMMARY Software product-line engineering is the successful reuse of technology when applied to component-based software development. The main concept and structure of this technology is developing reusable core assets by applying commonality and variability, and then developing new software using these core assets. Recently, the emergence of service-oriented environments, called SOA, has provided flexible reuse environments by reusing pre-developed component structure as service units; this is platform-independent and can integrate into heterogeneous environments. The core asset of an SOA is the service. Therefore, we can increase the reusability of an SOA by combining it with the concept of a product-line. These days, there exists research that combines SOA and product-lines, taking into account reusability. However, current research does not consider the interaction between the provider and consumer in SOA environments. Furthermore, this research tends to focus on more fragmentary aspects of product-line engineering, such as modeling and proposing variability in services. In this paper, we propose a mechanism named 2-Level SOA, including a supporting environment. This proposed mechanism deploys and manages the reusable service. In addition, by reusing and customizing this reusable service, we can develop and generate new services. Our proposed approach provides a structure to maximize the flexibility of SOA, develops services that consider systematic reuse, and constructs service-oriented applications by reusing this pre-developed reusable service. Therefore, our approach can increase both efficiency and productivity when developing service-oriented applications.

key words: SOA, service reuse, service variability, UDDI, service oriented development

1. Introduction

Component-based environments [1], [2] are changing to service-oriented environments [3]–[5], which develop new systems using services that are an abstraction of component units. In other words, heterogeneous components of executable code, for example using JAVA and C#, are abstracted as web services, called WSDL [6]. Therefore, in a service-oriented environment, we propose a flexible structure that makes the development of service applications possible through connection to and invocation of WSDL, which does not affect the kinds of component types that are implemented in heterogeneous environments.

With the advent of service-oriented environment, the interaction model called SOA [7], [8], based on UDDI [9] has been proposed. It enables the registration, discovery, and invocation of services using UDDI. In these SOA, the core element is the service. However, when the needed service is found by UDDI, according to a change of requirements and the service itself, such as name, operation, and parameters, the service is frequently redeveloped, not reusing the original service. Generally, service-oriented development in SOA is weak in changes to services. As a result, much time is consumed in analysis, design, and development of a new service when a change occurs. To overcome this weakness in service development and to increase reusability, trials have been conducted in which SOA is combined with product-line technology [10]–[12].

Product-line engineering [13], [14] is a technique that analyzes, identifies, and manages the commonality and variability [15] in a set of similar systems. As shown in Fig. 1, it consists of domain engineering that identifies and acquires commonality and variability in the domain containing a set of similar systems, decision and pruning that reuse the commonality and customizes the variable part of the system by deciding on the variability and application engineering that produces and develops new applications using this customized asset.

However, the approaches proposed in current research studies focus mainly on the domain engineering aspects [16]–[18], such as using the variability of the product-line concept for unit services or the composition of these services. In addition, it does not consider the SOA interaction mechanism that consists of a service consumer, a provider, and a service registry, called UDDI. In other words, we need an approach and an environment for developing service-oriented applications that are more systematic, flexible, and include reusability while considering SOA by combining product-line concepts.
In this paper, we propose a 2-Level SOA mechanism based on product-line engineering. The 2-Level SOA approach registers and generates reusable services that apply commonality and variability, and reuses this asset when generating new services. The domain engineering aspect of 2-Level SOA considers changes to services and supports the mechanism of registration and generation of a reusable service called vWSDL. With respect to decision and pruning, it supports the decision of the variability of this reusable service. With respect to application engineering, the customize services are registered and searched. Using this customized service, a new service-oriented application is developed. Therefore, this 2-Level SOA mechanism, including a supporting environment, can provide a seamless reusable development environment for developing service-oriented applications. The paper is organized as follows. Section 2 introduces our concept and definitions. Sections 3, 4, and 5 present the mechanisms of domain engineering, decision and pruning, and application engineering for 2-Level SOA with an implemented support tool, and a case study. Section 6 compares our method with related work. Section 7 concludes the paper and discusses future work.

2. Definition

2.1 Service with Variability

To support change to services in SOA, we use the concept of a service with variability [19], [20]. This concept denotes a reusable service that can respond to a change in a flexible way by designing services, operations, and messages by analyzing similar services that are provided in a domain. It has a variability type called a common/optional, an operation type, and message type.

-Common/optional variability means a service can be either common or optional.
-Operation type variability means this generalized operation type is specialized to many executable operations, both common and optional.
-MESSAGE type variability means a message can consist of one type element (simple), or at least two type elements (complex). Figure 2 shows the metamodel of a service with variability.

2.2 vWSDL Schema

We define vWSDL as an XML representation of a service with variability. To define this vWSDL specification, we designate the schema metamodel as shown in Fig. 3. In addition, Table 1 shows the elements and meanings of vWSDL. The service’s main element is the service itself, which includes operation, and messages. Therefore, vWSDL contains the key elements of variability. In addition the design of vWSDL takes the elements of WSDL into account.

General services are described and utilized using WSDL. WSDL does not, however, contain and explain the variability of service. Therefore, we define vWSDL, which does contain and explain the variability of service. vWSDL is a reuse model for generating a specific WSDL for a service. Using this defined vWSDL, a developer can specify and present the service with variability. In addition, by deciding and pruning the variability of vWSDL, a new specific WSDL can be generated easily.

2.3 vUDDI (UDDI of Service with Variability)

A registry of services with variability, called vUDDI, consists of publisherInfo, which contains information about the publisher of the service with variability, vModel, which represents the variability information model, including vWSDL, vModel key, etc., CategoryInfo, which is the category information of the registered vModel, and Opspec, which contains information about the vWSDL operation that is not realized. Figure 4 shows the elements of vUDDI.

In SOA, UDDI is used to discover and manage services. However, the current proposed UDDI cannot support the discovery and management of a service with variability. For this purpose, we therefore defined and developed vUDDI, which is based on UDDI.

As shown in Fig. 4, PublisherInfo represents the publisher information of vWSDL, such as ID, Name, Password, and so on. Table 2 explains the vModel element.
Table 1  vWSDL elements and meanings.

| vWSDL metamodel element | Meaning |
|-------------------------|---------|
| vService                | represents service with variability |
| vOperation              | denotes operation of service with variability |
| vMessage                | explains messages of service with variability |
| ServiceTypeVariability  | (c|p) represents the contained vService variability itself; c denotes it is common, p denotes it is optional |
| OperationTypeVariability| (c|p|c|p|a|p|a) represents the variability type of operation. c: this operation is a common type, which implies that it should be included. p: this operation is an optional type, which implies it may be included. ca: this operation is a common type and has an alternative variant operation. pa: this operation is an optional type and has an alternative variant operation |
| Alternatives            | group and represent the variant operations that have alternative relationships |
| endpoint                | represents the wsdl; if the service does not implement an operation, it denotes the OpSpec that provides the operation information |
| isDefault               | represents an operation that should be included among variant operations that have an alternative relationship |
| virtualName             | represents a variant operation that is not implemented |
| vCardinality            | denotes how many variant operations can be selected |
| MessageTypeVariability   | (c|p) denotes the variability type of an individual message; c denotes common message type that should be included; p denotes an optional message type that may be included |

Table 3 shows the OpSpec elements.

Based on this vUDDI structure, the consumer of the service with variability queries the vUDDI to find a vWSDL that can be reused to develop a new service. The query condition can be name, key and category. Using this vUDDI, we can construct and combine Product Line concepts in the SOA mechanism. In addition, it can support the generation and reuse of services with variability.

2.4 2-Level SOA Mechanism Based on Product-Line

Figure 5 explains the 2-Level SOA mechanism based on product-line engineering used to develop new services. It reuses vWSDL, which is a service with variability, and develops service-oriented applications reusing this customized service. There are three types of stakeholders that explain this mechanism.

- the provider of the service with variability: this stakeholder designs and develops the service with variability.
- the consumer of the service with variability: this stakeholder generates a new customized service, which is decided on variability of the service with variability.
- the service consumer: this stakeholder develops service-oriented applications by using and searching this customized service once the variability is decided.

As shown in Fig. 5, the 2-level SOA mechanism based on product-line engineering consists of a domain engineering phase, a decision and pruning phase, and an application engineering phase. The domain engineering phase generates and manages the service with variability through interaction with the vUDDI. The decision and pruning phase decides on the variability. The application engineering phase registers this customized service. Figure 6 shows the implemented...
In the following sections, we explain the domain engineering, decision and pruning, and application engineering phases for the 2-Level SOA mechanism using a case study and supporting tool. The case study uses a travel agency domain [21] that makes arrangements for people's holidays and journeys.

### 3. Domain Engineering for 2-Level SOA

#### 3.1 Specification and Realization of Service with Variability

The provider of the service with variability specifies the reusable service by analyzing the commonality and variability [15], [21] of a set of similar services in a domain. Table 4 shows an analysis of the result for services in a travel-agency domain that provides a travel reservation service (o indicates that the operation is provided and x indicates that the operation does not provide).

Figure 7 shows the reservation service (with variability) of the travel agency-domain.

As shown in Fig. 7, <<c>> denotes a common operation and <<p>> denotes an optional operation. If alternative operations exist that can replace these, then this operation is generalized and grouped. For example checkRealTimeInfo is a common type of generalized operation that has variant operations, named checkHotels and checkAirlines that can be replaced. In addition, the figure shows that a maximum of two elements can be selected.

After specifying and designing the service with variability, the provider of the service with variability constructs a service component that realizes this service. The realizing approach for service components can be divided into an all-in-one type and a separation type [22]. An all-in-one type realizes the variant operation as only one service component. The separation type realizes the variant operation as a large number of separated components. The separation
Table 4  Analysis of results for a travel-agency reservation service.

| Operation | Travel-Agency A Reservation Service | Travel-Agency B Reservation Service | Travel-Agency C Reservation Service | Operation Type | Variability |
|-----------|-------------------------------------|-------------------------------------|-------------------------------------|----------------|-------------|
| Reservation confirmation and cancelation (confirmTravelSchedule) | 0 | 0 | 0 | common: | 00> |
| Rent-Car Reservation (reserveRentCar) | x | x | 0 | optional: | 00> |
| Searching possibility of real time airline reservation (checkAirlines) | 0 | 0 | 0 | common: | 00> | Generalized operation: searching real time package (checkRealTimeInfo) | Variant: | 01... |
| Searching possibility of real time hotel reservation (checkHotels) | 0 | 0 | 0 | common: | 00> | Generalized operation: searching real time package (checkRealTimeInfo) | Variant: | 01... |
| Providing exchange rate Information (reportExchangeReport) | x | 0 | x | optional: | 00> | Generalized operation: providing travel destination information (reportTravelInfo) | Variant: | 01... |
| Providing time difference information (reportTimeLag) | 0 | x | x | optional: | 00> | Generalized operation: providing travel destination information (reportTravelInfo) | Variant: | 01... |
| Providing weather Information (reportWeather) | x | 0 | x | optional: | 00> | Generalized operation: providing travel destination information (reportTravelInfo) | Variant: | 01... |

Fig. 7  Reservation service with variability.

Fig. 8  Reservation service component diagram.

Fig. 9  Structure of vWSDL.

Fig. 10  vWSDL and Service component.

3.2 Registration of Service with Variability

The provider of the service with variability develops vWSDL to support a change in the service by analyzing services that have similar functionality in a domain, but have differences within an application. Here, vWSDL is a generalized reusable service that has common, optional, and alternative aspects. As shown in Fig. 9, vWSDL consists of one or more endpoints that reference services providing variant operations. Each endpoint denotes WSDL or OpSpec. If a variant operation is realized, then the endpoint specifies the WSDL. Otherwise, the endpoint denotes the OpSpec to implement a variant operation using this specified variant operation information, such as input, output, etc.

Figure 10 shows an instance of the vWSDL concept and structure.

Therefore, publishing and registering the service with variability consists of registering of the provider of service with variability, and publishing the vModel that describes the technical information of vWSDL and opSpec registra-
tion when the variation operation is not realized. Figure 11 shows the user interface of vWSDL registration. As shown in Fig. 11, the message, variant operation, and endpoint are published based on the schema proposed in Sect. 2.2. This figure shows the registration of the vWSDL of the reservation service with variability.

4. Decision and Pruning for 2-Level SOA

The consumer of the service with variability decides to generate a new service by reusing a vWSDL discovered in vUDDI.

As shown in Fig. 12, the retrieved vWSDL contains a variability type (vpType) about common, optional, and alternative variant operations. Therefore, the consumer of the service with variability prunes and decides on the variability based on this variability type, vpCardinality which denotes the range of selections, and the default property that should be included. Stereotype <<c>> is a common variant operation, which means it should be included. Therefore, with the exception of this common vpType decision, the variability pruning is divided into two types, namely an optional vpType decision and an alternative vpType decision.

-Optional vpType decision: Stereotype <<op>> denotes optional variability. For example, as shown in Fig. 12, op4 and op5 are optional variant operations, which mean they may be included. Therefore, we classify these optional selection decisions as optional vpType decisions.

-Alternative vpType decision: Stereotype <<ca>> means it is a common vpType, and so should be included. However, there are alternative variant operations that can be used. In addition, <<pa>> denotes that it is an optional vpType with alternative variant operations. Therefore, in the case of an alternative relationship, there is a given range of variant operations. In other words, in the case of <<ca>>, a minimum of one of variant operation should be selected. Other case of <<pa>>, none of them may be selected. Furthermore, in the case of <<ca>>, one of them is denoted as the default property, which means that this variation operation should be selected. Therefore, we define the variability decision considering the default property and the range of selection as alternative vpType decisions.

Figure 13 shows the vWSDL search from vUDDI. As shown in Fig. 13, key, name and category searching are provided. The vWSDL information is displayed at the bottom...
of the interface in the section titled vModel. Clicking the vWSDL button displays the discovered vWSDL information.

The consumer of the service with variability decides to generate a new customized service using the retrieved vWSDL. Figure 14 shows the variability decision of the vWSDL. As shown in Fig. 14, reserveRentCar is selected as an optional vpType decision. In addition, as an alternative vpType decision, checkRealTimeInfo with a common property and selection boundary of 1 to 2, is selected.

5. Application Engineering for 2-Level SOA

In application engineering for 2-Level SOA, using this decided and customized vWSDL, the consumer of the service with variability generates a new WSDL, including this se-
lected variant operation, and then registers it in the general UDDI. In addition, the service consumer generates a new service-oriented application using this generated service. Therefore, our approach provides a seamless and reusable life cycle that consists of generating new customized services by reusing service with variability and developing new service-oriented applications by reusing these customized services. Using the proposed mechanism, we can develop variant service-oriented applications more efficiently. As shown in our case study, we can develop various types of variant reservation services by reusing the vWSDL reservation element. Table 5 shows the variant services.

As shown in Table 5, a total number of 32 variant services can be developed using vWSDL.

6. Related Work and Evaluation

In this paper, we propose an SOA mechanism using UDDI and based on product-line engineering for creating and managing a vWSDL that applies commonality and variability, and for developing new service-oriented applications based on this mechanism. In addition, there is existing research on web service management based on pattern [16], adaptable service [17], VxBPEL [18] that deals with variability in BPEL [23], SORE [10], and managing SOA system variation [11].

This research proposed combining product-line engineering concepts. Web service management based on pattern [16] research focuses on the reusability of a web service. This research explains variability from three aspects, namely a WSDL document, a service endpoint, and business logic. In addition, it proposes variability management using patterns. However, this research does not include a detailed approach for explicitly specifying and deciding on variability. Adaptable service [17] proposes four types of variability, named workflow, composition, interface, and logic. This approach focuses on composing unit services and adapting services to react using an adaptable mechanism. VxBPEL [18] proposes an extension to the BPEL language, allowing for variability. However, it focused on changes to the composition of services. In addition to running VxBPEL, it needs a customized engine to interpret this extended variability. In [10], the role of service-oriented requirements engineering (SORE) as a mediator between application engineering and service engineering is discussed. However, this research study introduces only the concept and principles of SORE. In [11], a system for managing SOA variation is proposed, which comprises a combination of business process lines and process-oriented development. However, this research focuses more on the process variant aspects. Table 6 compares our approach to that in related work. As shown in Table 6, the related work that considers service-oriented product-lines does not fully support the entire core concept of product-lines in service aspects that consider the SOA mechanism, namely the development of reusable service assets and of customized service assets that reuse these reusable service assets. Here, we have particularly considered the interaction mechanism in SOA using UDDI. In addition, we provide seamless reusable environments for developing service-oriented applications. By generating reusable assets, named vWSDL, and using these reusable assets, we develop new customized services. These customize services are reused to develop service-oriented applications. We have a virtuous circle, based on vWSDL. We create and register new customized services by deciding on the variability according to requirement changes. Therefore, the customized service that is matched to the requirements is seamlessly created and registered in the UDDI, the needed service is found and used more often, and reusability is increased.

In addition, we have applied our product line-based approaches in several domains in which commonality and variability can be identified, such as shipping terminal operation system (TOS)’s berth planning [19] supply chain [20], hotel, movie, facility management, travel agency [21], smart shelf [24], etc. For each case study, we organized two development groups, each consisting of two or three persons, to measure the estimated effort required by our approaches in comparison with that required by other approaches. The members of these development groups have the same educational background and very similar programming experience. One group used our proposed approach that constituted an extended 4D process [21], [24] mechanism; the other group did not use our approach, but rather general development approaches that consist of a design, an analysis, and an implementation phase. In these experiments, we did not count the training period or the time consumed explaining our approaches to the groups that used our proposed approach. Table 7 shows the estimated average time for developing a single service-oriented application. As shown in Table 7, in the first development of a new service-oriented application, Group A spent much more time than Group B, because a lot of time was consumed analyzing the commonality and variability of services. We present the results of the estimation using a line graph as shown in Fig. 15. Group B started developing the application without using the reuse approach. This definitely saved a lot of development time.

| Table 5 | Variant services. |
| --- | --- |
| **Reservation vWSDL operation variability** | **Variant WSDL (included operation)** | **Variant service case** |
| <<<<>> | One (`C1`) | common: 2types |
| [1..2]checkRealTimeInfo | `default` | common+optional: 2types |
| | `therefore` | common+optional- alternative: 14types |
| | only | common+optional+optional- alternative: 14types |
| | checkHotels is selected | |
| two (`C2`) | |
| <<>> | None (`C3`) | |
| [0..1]reserveRentCar | Select (`C1`) | |
| <<>> | None (`C3`) | |
| [0..3]reportTrainInfo | One (`C1`) | |
| | Two (`C2`) | |
| | All (`C2`) | |
### Table 6  Comparison with related work.

| Item             | Web Service Management based on pattern [16] | Adaptable Service [17] | VxBPEL [18] | Managing SOA System Variation [11] | 2-Level SOA (Our approach) |
|------------------|---------------------------------------------|------------------------|-------------|-----------------------------------|----------------------------|
| Goal             | To support the reusability in web service development | To support adaption of services | To support changes in service composition | To permit the selection of the suitable process variant and implement the appropriate SOA system | To support the development of service and service-oriented applications using the systematic and seamless reusability concept |
| Characteristic   | Proposes web service management using patterns | Proposes an adaptable service model | Extends BPEL to represent variability | Proposes process-oriented development | Using SOA mechanism and UDDI based on product-line engineering concept |
| Reflection of variability | Only proposes the variability type | Proposes the variability type and focuses on designing adaptable services | Focuses on variability within service composition aspects | Focuses on variability in actions in business processes | Proposes the vWSDL, which reflects variability |
| Representation of variability | Does not provide an explicit representation method | Explains, using examples but does not provide well-organized elements to represent variability | Represents extended elements using BPEL language | Explains, using examples, but does not provide well-organized elements to represent variability | Provides explicit elements to represent variability and provides vWSDL based on XML representation |
| Decision of variability | Not provided | Provides a decision model, but decides using the adaptation mechanism, not using the variability decision | It is not a variability decision, but rather a change of process composition | Provides variability selection table | Provides decision based on variability type |
| Life cycle       | Supports design | Supports design | Supports execution | Supports design | Supports design and execution |
| Tool support     | Provides a pattern modeling tool | Not provided | Prototype environment to interpret extended BPEL | Provides a tool to support detailed process model (DPM) generation | Provides an implemented SOA environment using UDDI to design, create and manage services with variability |

### Table 7  Estimated average period (1st round) – single service oriented application development.

| Phase | Task                                      | Days | Phase | Task                                      | Days |
|-------|-------------------------------------------|------|-------|-------------------------------------------|------|
| Analysis | Define domain knowledge                   | 14   | Analysis | Define requirement                       | 15   |
|       | Define domain context                     |      |       |                                           |      |
|       | Construct feature model                   |      |       |                                           |      |
|       | Identify domain business process activity |      |       |                                           |      |
|       | Group and generalize domain process activity | 13   |       |                                           |      |
|       | Identify business process scenario         |      |       |                                           |      |
|       | Model business process with variability   |      |       |                                           |      |
| Design | Identify candidate operation type         | 8    | Design | Design service model                     | 17   |
|       | Group and generalize candidate operation type |      |       |                                           |      |
|       | Define message type                       |      |       |                                           |      |
|       | Construct extend feature model            |      |       |                                           |      |
|       | Model specification level of service with variability | 10   |       | Specify service specification            |      |
|       | Define specification of service with variability |      |       |                                           |      |
|       | Refine composition scenario               | 12   |       |                                           |      |
|       | Model composition level of service with variability |      |       |                                           |      |
| Development | Select realization mechanism              | 33   | Development | Develop service                         | 30   |
|       | Generate service component model          |      |       |                                           |      |
|       | Develop service component                 |      |       |                                           |      |
|       | Develop service                           | 19   |       | Develop service application              |      |
|       | Generate WSDL                             |      |       |                                           |      |
|       | Generate BPEL                             |      |       |                                           |      |

Group A: 109 work days  
Group B: 62 work days
It may have taken Group A more time to develop the application because the commonality and variability analysis is complicated. In the second round of development (number of application, 2), Group A's development time was almost the same as Group B’s. In the third round of development (number of applications, 3), Group A’s development time was shorter than Group B’s, because they could reuse the pre-developed vWSDL. Using this vWSDL, they develop different type of applications efficiently.

As shown in this graph (Fig. 15), once the reusable artifact (vWSDL) has been developed, applications derived from it can be easily and quickly developed. That is, the greater the number of similar applications developed in a domain, the greater is the effect induced.

7. Conclusion and Future Work

The service-oriented paradigm provides for a flexible environment to integrate heterogeneous system using services as a more abstracted concept. The reusability aspects are combined in this flexible environment, which means that the benefit of SOA is maximized. However, current SOA environments do not have these advantages. As in component-based development, when a new service is needed, or a service changes, the tasks of analysis, design, and development are repeated. To solve these problems and to support systematic and efficient design and development, we propose an integrated 2-Level SOA mechanism and supporting environment using UDDI based on the product-line concept. Through this mechanism, a systematic seamless and reusable environment is obtained from design to the development of the service and service-oriented application. This increases efficiency and decreases costs when developing service-oriented applications. In future work, we plan to research how often reusable assets are used and an efficient approach to processing queries for reusable and managed assets.

Acknowledgements

This work was supported by the National Research Foundation of Korea Grant funded by the Korean Government (MEST) (NRF-2010-2100328000). The authors would like to thank the members of software engineering lab in Pusan National University: Baek Sun-jae and Shin Soo-Hye for supporting this project.

References

[1] G. Pour, “Component-based software development approach: New opportunities and challenges,” Technology of Object-Oriented Languages TOOLS 26, pp.376–383, Aug. 1998.
[2] S. Mahnood, R. Lai, and Y.S. Kim, “Survey of component-based software development,” IET Software, vol.1, no.2, pp.57–66, April 2007.
[3] M.P. Papazoglou and W.J.V.D. Heuvel, “Service-oriented architectures: Approaches, technologies and research issues,” International Journal on Very Large Data Based, vol.16, no.3, pp.389–415, July 2007.
[4] T. Erl, SOA Principles of Service Design, Prentice-Hall, 2008.
[5] J. Park, M. Moon, and K. Yeom, “The BCD view model: Business analysis view, service Composition view and service Design view for service oriented software design and development,” Future Trends of Distributed Computing Systems, pp.37–43, Oct. 2008.
[6] W3C, “Web Service Description Language (WSDL) Version 2.0 Part 0; Primer,” http://www.w3.org/TR/2007/REC-wsdl20-primer-20070626
[7] M. Endrei, J. Ang, A. Arsanjani, S. Chua, P. Comte, P. Krogdahl, M. Luo, and T. Newling, “Patterns: Service-Oriented Architecture and Web Services,” SG24-6303-00, IBM Redbooks publication, http://www.redbooks.ibm.com/abstracts/sg246303.html?Open, April 2004.
[8] M.N. Huhns and M.P. Singh, “Service-oriented computing: Key concepts and principles,” IEEE Internet Computing, vol.9, no.1, pp.75–81, Jan–Feb. 2005.
[9] OASIS, Universal Discovery, Description and Integration (UDDI) v3.0.2, http://www.oasis-open.org/specs/index.php#uddi\v3.0.2
[10] S. Adam and J. Doerr, “The role of service abstraction and service variability and its impact on requirement engineering for service-oriented systems,” International Conference on Computer Software and Applications, pp.631–634, July 2008.
[11] N. Boffoli, M. Climitele, F. Maggi, and G. Visaggio, “Managing SOA system variation through business process lines and process oriented development,” International Software Product Line Conference, Aug. 2009.
[12] E. Maruguesupilai, B. Mohabbati, and D. Gasevic, “A preliminary mapping study of approaches bridging software product lines and service-oriented architectures,” International Software Product Line Conference, Aug. 2011.
[13] P. Clements and L. Northrop, Software Product Lines Practices and Patterns, Addison-Wesley, 2002.
[14] K. Pohl, G. Böckle, and F.V. Linden, Software Product Line Engineering foundations, Principles, and Techniques, Springer, Berlin Heidelberg New York, 2005.
[15] M. Moon, K. Yeom, and H.S. Chae “An approach to developing requirement as a core asset based on commonality and variability analysis in a product line,” IEEE Trans. Softw. Eng., vol.31, no.7, pp.551–569, July 2005.
[16] J. Jiang, A. Ruokonen, and T. Systa, “Pattern-based variability management in Web service development,” Third IEEE European Conference on Web Services, pp.12–23, Nov. 2005.
[17] S. Chang, H. La, and S. Kim, “A comprehensive approach to service adaptation,” International Conference on Service Oriented Computing and Applications, pp.191–198, June 2007.
[18] M. Koning, C. Sun, M. Sinnema, and P. Avgeriou, “VxBPEL: Supporting variability for Web services in BPEL,” Information and Software Technology, vol.51, no.2, pp.258–269, Feb. 2009.
[19] J. Park, J. Kim, S. Yun, M. Moon, and K. Yeom, “An approach

Fig. 15 Estimated efforts for developing service-oriented applications.
to developing reusable domain services for service oriented applications,” ACM Symposium on Applied Computing, pp.2252–2256, March 2010.

[20] J. Park, M. Moon, and K. Yeom, “Variability modeling to develop flexible service-oriented applications,” J. Systems Science and Engineering, vol.20, no.2, pp.193–216, June 2011.

[21] J. Park, Development of Services with Variability to Construct an SOA-based Product Line, PhD thesis, Pusan National University, 2012 (in Korean).

[22] J. Park, M. Moon, and K. Yeom, “An approach to realizing a service with variability for service oriented applications,” J. KIISE: Software and Applications, vol.38, no.2, Feb. 2011 (in Korean).

[23] OASIS, Web Services Business Process Execution Language Version 2.0, http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html, April 2007.

[24] J. Park, M. Moon, T. Nam, and K. Yeom, “A 4D process for service oriented software development,” J. KIISE: Software and Applications, vol.35, no.11, Nov. 2008 (in Korean).

Joonseok Park received the bachelor’s degree in Computer engineering from the Pukyong National University in 1999 and the MS degree in 2002 and the Ph.D. degree in 2012, both in computer engineering from Pusan National University. Currently, he is a Post-doctoral Researcher in the BK21 Center for U-Port IT Research and Education at the Pusan National University. His research interests include service oriented computing, ubiquitous computing, software architecture, and product-line engineering.

Mikyeong Moon is an associate professor of computer and information engineering at the Dongseo University in Busan, KOREA. She received the BS degree in 1990 and the MS degree in 1992, both in computer science from the Ewha Womans University, Seoul, Korea, and the Ph.D. degree in computer science and engineering from Pusan National University in 2005. Her current research interests include software reuse, product line engineering, requirement engineering, and RFID solutions. She is a member of the Korea Information Science Society.

Keunhyuk Yeom received the bachelor’s degree in computer science and statistics from Seoul National University in 1985 and the MS and Ph.D. degrees in computer and information science and engineering from the University of Florida in 1992 and 1995, respectively. He is professor of computer science and engineering at the Pusan National University in Busan, Korea. From 1985 to 1990, he was with LG Electronics Research Institute. His research interests include software product lines, component-based software development, software reuse, self-adaptive software, situation-aware framework for RFID/USN Solutions, etc. He is a member of the ACM, the IEEE Computer Society, and the Korea Information Science Society.