Practical Methods for Adapting Services Using Enterprise Service Bus*

Hyun Jung La, Jeong Seop Bae, Soo Ho Chang, and Soo Dong Kim

Department of Computer Science
Soongsil University, Seoul, Korea
511 Sangdo-Dong, Dongjak-Ku, Seoul, Korea 156-743
{hjla, jsbae, shchang}@otlab.ssu.ac.kr, sdkim@ssu.ac.kr

Abstract. In service-oriented computing (SOC), services are designed not just for a dedicated client but for a family of potential clients. For services to be generic and serviceable to different clients, service variability among the clients must be analyzed and modeled into service components. *Enterprise Service Bus (ESB)* is an architectural framework for service integration, but it does not provide effective adaptation mechanisms. Hence, it is desirable to devise techniques to adapt services on ESB for specific service requests. In this paper, we identify four types of service variability, and we present methods to adapt services provided on ESB. These methods can be practically applied in designing highly adaptable services on ESB.

1 Introduction

In service-oriented computing (SOC), services are designed not just for a dedicated client, but for a family of potential clients. A key problem in developing such common services is to model the variability embedded in common features, and to be able to adapt common services for specific service requests [1][2]. Another case requiring service adaption is when there is a *partial matching* between available services and services expected by clients. That is, an available service can potentially fulfill the service expected by a client, but they do not match in full [3]. Hence, identifying types of variability which may occur on services and how service variability can be modeled into adaptable services are two essential prerequisites to designing highly reusable and applicable services.

Enterprise Service Bus (ESB) is an architectural framework to integrate heterogeneous services and applications in distributed network [4]. It provides the integration functionality through transformation, communication, and routing. Beyond this, however, ESB does not provide methods for service adaptation. Hence, it is desirable to devise techniques to adapt services on ESB for specific service requests.

In this paper, we first identify four types of variability which may occur on services. Then, we present adaptation methods for services on ESB; *Workflow Mediator, Service Binder, Interface Transformer,* and *Logic Broker*. Each method is presented

* This work was supported by the Korea Science and Engineering Foundation (KOSEF) grant funded by the Korea government (R01-2005-000-11215-0 (2007)).

L. Baresi, P. Fraternali, and G.-J. Houben (Eds.): ICWE 2007, LNCS 4607, pp. 53–58, 2007.

with its overall scheme and a specific adaptation algorithm based on ESB specification. These methods can be practically applied in designing highly adaptable services on ESB.

2 Related Works

Sam's work proposes a customization framework for dynamic Web services [5]. Based on the comparison to syntactic/semantic aspects and to the constraint on input and output, it suggests a matchmaking process and conflict resolution mechanism for service adaptation. However, this work deals with only interface adaptation. Jiang's work proposes a categorization of variation points and introduces a pattern-based approach for managing the variation points [6]. However, instructions for identifying variability and adaptation are not included. Herault's work proposes an approach to mediating and executing operations on ESB [7]. It explains how mediation can be achieved on ESB. However, how they mediation is performed is not given in enough details. Schmidt's work addresses a mediation model on ESB, which reconfigures the links between bus service providers and requesters to create dynamic alternations to routing and to modify their behaviors [8]. It treats interface mediations and policy mediations. However, practical adaptation mechanisms for these are not given.

From our survey, we observe that current research works treat service adaptation at conceptual level rather than design level, and adaptation methods on ESB treat only interface and logic variability. In this paper, we treat four types of service variability and suggest design-level practical methods for adapting services on ESB.

3 Types of Service Variability

In this section, we identify four places where service variability may occur [9].

Workflow Variability. A business process consists of a sequence of activities, i.e. *unit services* [10], and this sequence is called a *workflow*. For a given business process, the workflow may slightly vary, depending on different service clients. That is, some unit services in a workflow may be optional, and there can be more than one execution path for the given workflow.

Composition Variability. Services are discovered at runtime, and there may be more than one unit service which fulfills the required functionality. In this case, variation occurs on binding the right services. That is, for each specific request, one of the candidate unit services must be composed.

Interface Variability. *Variability on interfaces* occurs when the interfaces of unit services do not match to the interfaces of published services. Even if the functionality of a unit service is met by the functionality of a registered service, the signatures of interface and the semantics may not fully match. This should be modeled during service engineering, and the mismatch should be resolved by some interface adaptation mechanism.

Logic Variability. Service component includes operations for providing service functionality. There may be minor variation on the business logic or algorithm for the service component. This micro-level logic variability should be modeled into a service component, so that it can be tailored for each invocation.

4 Adaptation Managers for Services on EBS

In SOC, service clients do not have access to the internal details of services, and hence adaptation of a service is performed by an external software agent, which we call it *adaptation manager*. We propose a general scheme of designing ESB-based adaptation mangers which can be used in resolving all four types of variability.

To implement adaptation managers on ESB, we define two kinds of components; *listener* and *adaptor*. As shown in Figure 1, a *listener* with «Listener» stereotype listens and intercepts service requests made by clients. And, it determines the required adaptation for each invocation, and invokes appropriate *adaptors*. The *adaptors* actually perform the requested adaptation over *service components* through *end-points* of ESB. Figure 1, shows four types of adaptors, which are denoted with «adaptor» stereotype.

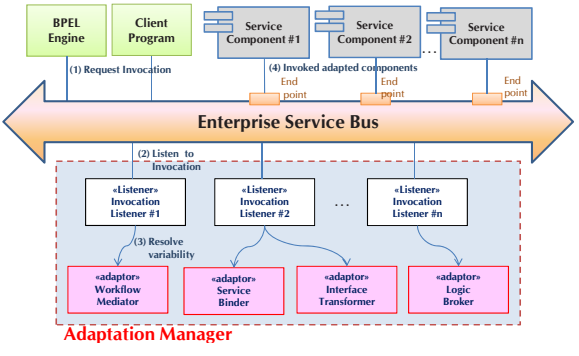

Fig. 1. Adaptation Manager deployed on ESB

4.1 Workflow Mediator for Workflow Variability

Workflow Mediator is to adapt the workflow of services, and it is implemented with *Mediator* pattern. When modeling workflow variability, variation points of workflow type and their relevant variants are identified and stored in the *workflow repository*.

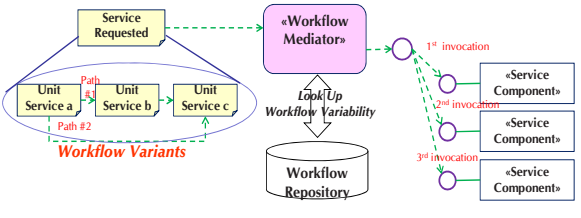

Fig. 2. Pattern of Workflow Mediator

As shown in Figure 2, the workflow mediator analyzes service requirements using user's preference and context information, determines an appropriate *workflow variant* from the repository, and makes a series of invocations over the participating service components by using the rule repository.

We define two adaptation methods for workflow mediators as shown in Figure 3; *determineWorkflow()* and *conctroServicelInvocation()*. The *determineWorkflow()* method is to select the most appropriate workflow from the repository based on the service requirements. This operation returns a path of the BPEL specification which will be executed. The *conctroServicelInvocation()* method is to execute the workflow by invoking appropriate service components.

```
public String determineWorkflow(MessageExchange exchange) throws MessagingException {

    String BPELPath; // path of the BPEL documents
    Message sourceMsg = getSourceMsg(exchange);    // to get source invocation message
    Vector workflowVariant = getWorkflowVariants(exchange); // to get  candidate workflows

    // to get a particular workflow among candidate workflow based on the condition
    for ( int i = 0 ; i < workflowVariant.size() ; i++ ) {
        if (compare (workfowVariant[i], condition) == true) {
            BPELPath = workflowVariant[i].path;
            return BPELPath;
        }
    }
}

public void controlServiceInvocation(MessageExchange exchange) throws MessagingException {

    Message sourceMsg = getSourceMsg(exchange);    // to get source invocation message
    Vector serviceComp = getServiceComponents (exchange); // to get  the service components
    Rule rule = getRule (exchange ) // to get the rule for the exchange

    // to invoke a service component by comparing when the service component is invoked
    for ( int i = 0 ; i < serviceComp.size() ; i++ ) {
        if (compare (serviceComp[i].case, rule.case) == true)  CallComponents (serviceComp[i]
    }
}
```

Fig. 3. Algorithm of *Workflow Mediator*

4.2 Service Binder for Composition Variability

Service Binder is to adapt the service compositions, and it is implemented with *dynamic selection* pattern. When modeling composition variability, variation points of composition type and their variants are identified and stored in *composition repository*.

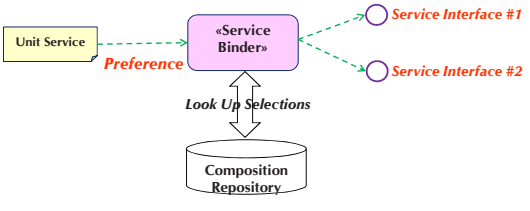

Fig. 4. Pattern of *Service Binder*

```
public String determineInterface (MessageExchange exchange) throw MessageException{

        Message sourceMsg= getSourceMsg(exchange); // to get source invocation message
        Vector  endPoints = getEndPoint(exchange);      // to get possible end points
        Rule rule = getRule (exchange);          //to get selection rule for the exchange
        String targetEndPoint;                   //to save a target end point

        // to get an appropriate end point
        for (int i=0;i<endPoints.size(); i++) {
            if (compare(endPoints[i].case, rule.case) == true)
                targetEndPoint = (String)endPoint[i];
        }
    }

public void bindInterface  (Endpoint ed) throw MessageException{
        if  (comparingInterface(sInvc, ed) == match)  call(ed);   // adaptation is not needed.
        else call(interfaceMediator, sourceMsg) // in order to adapt interface mismatch
}
```

Fig. 5. Algorithm of *Service Binder*

Figure 4 shows the relationship among the service binder, unit services, composition repository and WSDL service interfaces.

We define two methods for service binder; *determineInterface(),* and *bindInterface().* The *determineInterface()* method is to select an *interface variant* in the repository based on user preferences, characteristics of services, and other context, and to generate a service endpoint type which is specific to the interface variant. The *bindInterface()* method is to bind the specified WSDL interface to the unit service. If interface mismatch occurs, it should be resolved with *Interface Transformer.*

4.3 Interface Transformer for Interface Variability

Interface Transformer is to adapt interfaces, and it is placed between unit services and service providers. When modeling interface variability, variation points of interfaces and relevant interface variants are identified and stored in the *Interface Transform Repository*. Each transformation method takes a service invocation of the interface specified by a unit service, transforms it into the published interface which eventually maps to the interface of a service component. It analyzes the service innovation, determines a transformation method, performs the transformation, and generates a new service invocation of a published interface.

4.4 Logic Broker for Logic Variability

*Logic adaptor i*s a kind of adaption manager use the *plugin* method with *profiles* since the method makes the variable logics more decoupled [2]. When modeling logic variability, client profiles and objects which implement algorithms are identified and stored in *Client Profile* and *Logic Object Repository*.

Message listener recognizes the invocation from Web service client to the service interface written in WSDL. The listener invokes the *variability analyzer* so that the variability analyzer finds out corresponding adaptation types. Then, the listener invokes the service component with the object path through *object finder* so that the service component can use the logic object. For the logic variants of unknown and newly added unit services, new algorithm object should be deployed on the *object container*.

5 Conclusion

A key problem in developing reusable services in SOC is to identify the common features among potential clients and to model them into service components. In addition, *variability* within a common feature should also be modeled. Moreover, *partial matching* between available services and requested services should be identified can resolved. ESB provides an architectural framework for service integration, without providing practical adaptation mechanisms.

In this paper, we identified four types of variability which could occur on services. By extending software adaptation techniques and utilizing the key features of ESB, we presented practical methods to adapt services on ESB. Each method was presented with its overall scheme and a specific adaptation algorithm based on ESB specification. By using the methods, services with high adaptability and applicability on ESB can be effectively developed.

References

[1] Kim, S., Her, J., Chang, S.: A Theoretical Foundation of Variability in Component-Based Development. Information and Software Technology (IST) 47, 663–673 (2005)

[2] Chang, S.H., Kim, S.D.: A Systematic Approach to Service-Oriented Analysis and Design. In: the proceedings of the 8th International Conference on Product Focused Software Development and Process Improvement (PROFES) (to Appear)

[3] Min, H., Choi, S., Kim, S.: Using Smart Connectors to Resolve Partial Matching Problems in COTS Component Acquisition. In: Crnković, I., Stafford, J.A., Schmidt, H.W., Wallnau, K. (eds.) CBSE 2004. LNCS, vol. 3054, pp. 40–47. Springer, Heidelberg (2004)

[4] Chappell, D.A.: Enterprise Service Bus, O'Reilly (2004)

[5] Sam, Y., Boucelma, O., Hacid, M.: Web Services Customization: A Composition-based Approach. In: ICWE'06. proceedings of the International Conference on Web Engineering, IEEE Computer Society Press, Los Alamitos (2006)

[6] Jiang, J., Ruokonen, A., Syata, T.: Pattern-base Variability Management in Web Service Development. In: ECOWS '05. proceedings of the Third European conference on Web Services, IEEE Computer Society Press, Los Alamitos (2005)

[7] Herault, C., Thomas, G., Lalanda, P.: Meditation and Enterprise Service Bus A position paper. In: the proceedings of the First International Workshop on Mediation in Semantic Web Services (MEDIATE 2005), pp.1-14 (2005)

[8] Schmidt, M.T., Hutchison, B., Lambros, P., Phippen, R.: The Enterprise Service Bus: Making Service-oriented Architecture Real. IBM Systems Journal 4(4), 781–797 (2005)

[9] Chang, S.H., La, H.J., Kim, S. D.: A Comprehensive Approach to Service Adaptation, IEEE International Conference on Service-Oriented Computing and Applications (SOCA) (to Appear)

[10] OMG Business Process Management Initiative, Business Process Modeling Notation (BPMN) Version 1.0, OMG Final Adopted Specification (February 6, 2006)