

Service Supervision: Coordinating Web Services in Open Environment

Masahiro Tanaka¹, Toru Ishida^{1,2}, Yohei Murakami¹, Satoshi Morimoto²

¹Language Grid Project, National Institute of Information and Communications Technology,

²Department of Social Informatics, Kyoto University,

Kyoto 606-8501 Japan

3-5 Hikaridai, Seika-cho, Kyoto, Japan

{mtnk, yohei}@nict.go.jp, ishida@i.kyoto-u.ac.jp, morimoto@ai.soc.i.kyoto-u.ac.jp

Abstract

A composite Web service designed based on abstract Web services, which define only interfaces, allows an application developer to select services required for his application only by setting endpoints for the atomic Web services. In open environment, however, the composite Web service configured in this manner may fail due to unique behaviors of the selected services. It is difficult for the designer of the composite Web service to prevent the failure because he does not know which services are selected and how they behave. On the other hand, the application developer is not authorized to modify the composite Web service due to the need to protect intellectual rights. Our solution is Service Supervision, which monitors and controls execution of composite Web services. Service Supervision makes the followings possible. 1) An application developer can control the behavior of a composite Web service by changing the execution state, even if he is not authorized to modify the composite Web services. 2) A control pattern for coordinating Web services can be applied to various composite Web services in order to reduce the load imposed by designing control processes. In order to realize Service Supervision, we introduce meta-level control of a composite Web service. Moreover we then use the choreography to define the interaction protocols for the controls. The proposed framework is based on existing standard languages, WS-BPEL and WS-CDL. Therefore we can exploit existing tools and expertise of SOA engineers.

1 Introduction

Recently, various organizations have been publishing their programs or contents as Web services in the open en-

vironment. This makes it possible for anyone to combine the Web services provided by various organizations.

For example, programs for language processing such as machine translators and morphological analyzers, and contents like dictionaries and multilingual corpora are now available on Language Grid[7] as Web services. More than 30 programs and contents have been collected from various organizations like companies and research institutes. This leads to the creation of new composite Web services created by fusing the atomic Web services provided by various organizations. On the basis of the Web services and the infrastructure, some technologies have been developed such as context coordination between machine translation services[11] and integration with other infrastructure for natural language processing components[2].

Once many Web services whose interfaces are standardized based on service types become available, a composite Web service can be defined based on atomic Web service only whose interfaces are defined (we call them an abstract atomic/composite Web service). An application developer can select atomic Web service which is actually invoked (we call it a concrete atomic Web service) only by setting an endpoint for each abstract atomic Web service. Hassine et al. referred to this style of Web service composition as horizontal Web service composition[6]. Horizontal Web service composition will become more and more important as the number of Web services increases and standardization of Web service interfaces is achieved. The standardization also makes it possible to improve performance of communication between Web services based on contexts[8].

However, in open environment, there is a problem which have not been considered for development of information systems in one company or companies which work together. When an application developer performs horizontal Web service composition, the designer of the abstract composite Web service cannot know which concrete atomic Web services are assigned and how they behave. This may cause runtime failure due to unique behaviors of each concrete

*This work was done while the first author was a Ph.D student at Department of Social Informatics, Kyoto University.

atomic Web service which is not represented in WSDL documents defined for the abstract atomic Web service.

On the other hand, the application developer who selects concrete atomic Web services by setting endpoints may know details of the specification of the atomic concrete Web services. But he also has difficulty in coordinating the Web services to suit the behaviors for the following reasons.

No authority to modify composite Web service In the open environment, the application developer generally does not belong to the organization that created the abstract composite Web service that he configures. Therefore, the application developer may not be authorized to modify the abstract composite Web service under terms set to protect the intellectual rights of the creator of the composite Web service.

Load of adding processes for coordination Even if the application developer is authorized to modify a composite Web service, the user has to add processes to realize coordination. This burden is too heavy for the application developer.

Our solution is the concept of Web service coordination; instances of a concrete composite Web service, whose atomic services are assigned to concrete services, are monitored and controlled from outside the composite Web service. This does not demand modification of the original abstract composite Web service. We call this concept "Service Supervision". Service Supervision makes the following two points possible.

First, Service Supervision makes it possible to separate the control processes needed for coordination from the composite Web service. This allows the application developer to describe only those processes needed for the coordination without modifying the composite Web service.

Next, Service Supervision allows an application developer to create a general control pattern for coordination. Once created, the control pattern can be used against other composite Web services. This reduces the load of describing control processes for coordination.

Some previous works also aim at changing the behavior of a composite Web service at runtime by extending a standard language or the execution engine for composite Web services. Most of the works have adopted the concept of AOP (Aspect-oriented Programming). Some monitor and modify messages exchanged between services for service selection and error handling[9, 1]. Transparent transformation of a composite Web service has been also tried[10]. AO4BPEL[4] directly adopted AOP and allows us to weave a process described in BPEL into an existing BPEL process.

But these works have not focused some requirements for runtime control. First control process should be described based on the existing standard frameworks. This is because

there are already many tools compliant with some standards and engineers have learned a lot about the standards. Next flexibility of control is required for some types of controls. For example, weaving a process based on AOP does not realize control of execution of a composite Web service, such as stopping or restarting a process. But no previous work solved both of these points.

In this paper, we combine the following two approaches to realize Service Supervision. First we introduce meta-level control of a composite Web service, which monitors and changes the state of execution of a composite Web service. We can change the behavior of an active instance of a composite Web service and collect information for coordination through the meta-level control. Moreover, we define a protocol as choreography in order to control multiple instances of composite Web services. Our framework allows us to describe control based on existing standards, WS-BPEL¹ and WS-CDL² and to flexibly control execution of a composite Web service.

The rest of this paper is organized as follows. In Section 2, we explain the features of Web services in the open environment and the requirements for coordinating Web services by showing a typical scenario. We then details the concept of Service Supervision and the implementation based on the current standard languages, WS-BPEL and WS-CDL in Section 3. We first describe the coordination of atomic Web services in the open environment and meta-level control as the solution and then show how choreography is used for controlling the interaction of composite Web services. Section 4 discusses the applicability and the performance of our framework. After introducing related works in Section 5, we conclude this paper in Section 6.

2 Scenario

In open environment for Web service, such as the Language Grid[7], a Web service can be shared by some composite Web service as shown in Fig. 1. For this reason, an execution of a composite Web service may affect an execution of other composite Web service. Moreover, bindings from atomic Web services in a abstract composite Web service to concrete atomic Web services are determined based on requirements of application developer who invokes the composite Web service from his application. This is the reason the designer of the composite Web service does not know which concrete atomic Web services are actually invoked and how they behave.

Take the composite Web service in Fig. 2 as an example. The notation in the figure follows BPMN, a language for

¹<http://www.ibm.com/developerworks/library/ws-bpel/>

²<http://www.w3.org/TR/ws-cdl-10/>

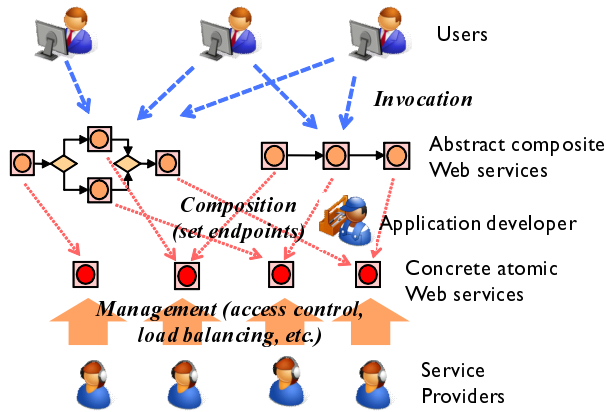


Figure 1. Web services in open environment.

modeling a business process. Circles at the both ends represent the start and the completion of the composite Web service. A rectangle which has a word inside represents an invocation of other Web service from the composite Web service. A rectangle with an arrow forming a round shape represents a block which is iteratively executed. The Web service invocations in the rectangle are executed while the given condition is satisfied. To simplify the figure, we omitted the processes of copying variables between input and output of the Web service, receiving a request, and returning a response from/to the client.

Several standard languages exist for describing composite Web services, such as WS-BPEL and OWL-S³. We can also combine Web services by writing programs in languages such as Java. However, we focus here on composite Web services as a business process described in these languages.

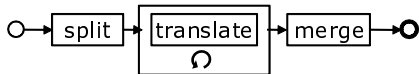


Figure 2. An example of a composite Web service.

The composite Web service shown in Fig. 2 translates document using a machine translation Web service. Our goal is to prevent the translation of an extremely long document from exceeding the limit of length set for the original machine translation Web service. This composite Web service first splits the given document into sentences (`split`). Next it translates the sentences by the machine translation Web service in the loop. Finally, it merges the results of the machine translation Web service and returns the translation result of the given document. The execution engine for

³<http://www.daml.org/services/owl-s/1.1/>

composite Web services creates an instance of the composite Web service when it is invoked by other Web service or a client application.

However, in the open environment, the execution of the document translation composite Web service shown in Fig. 2 may fail due to the runtime environment. One possible failure is the limit set by the provider of the machine translation Web service which is assigned to invocation “translate”. Assume that each user group that can access the machine translation Web service is assigned an ID, and that the provider of the machine translation Web service limits the number of invocations that each user group can make. In this case, the number of invocations can exceed the limit. Once the number of invocations exceeds the limit, all instances of the document translation composite Web service invoked by the users in the group fails.

We can solve these problems if we can modify the document translation composite Web service. One solution to the first case is to switch to a different machine translation Web service when the number of invocations approaches the limit. When the interfaces are standardized and support all machine translation Web services, it is simple to switch to another machine translation Web service just by changing the endpoint.

To realize this solution, we need to modify the document translation composite Web service as shown in Fig. 3.

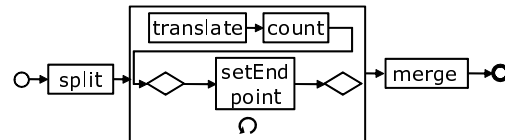


Figure 3. Control by modifying composite Web service.

The composite Web service shown in Fig. 3 uses an external service which records the number of invocations. Before invoking the machine translation service, the composite Web service invokes the external service to increment the recorded number of invocations (`count`). This external service returns the number of invocations. When the number of invocation approaches the threshold, the endpoint of the machine translation Web service is changed (`setEndpoint`). Figure 3 shows the conditional execution using two rhombuses. `setEndpoint` between them is executed only if the number of invocations exceeds the limit. An endpoint for each service invocation is represented as a special variable in a composite Web service. Therefore `setEndpoint` is implemented as a copy operation of a literal to the variable.

3 Service Supervision

In open environment, it is difficult to design the composite Web service shown in Fig. 3 because the designer does not know which concrete atomic Web services are assigned and how they behaves.

In this paper, we introduce meta-level controls of a composite Web service in order to change its behavior without modification of the definition. The meta-level controls make it possible to define a composite Web service which controls other composite Web service. The interaction between the composite Web service for the controls and composite Web services to be controlled is defined using a choreography. Based on the above approaches, we realize Service Supervision, which monitors and controls execution of a composite Web service without modifying the composite Web service.

In this section, we first describe the meta-level controls of composite Web services. Then we showed the architecture which uses the meta-level controls and choreography described in standard languages, WS-BPEL and WS-CDL.

3.1 Meta-level Control of Composite Web Service

In this paper, we introduce meta-level controls of a composite Web service in order to change its behavior without modification of the definition. Table 1 shows a list of meta-level functions. The functions are provided by Web APIs. This is the reason we can define a composite Web service which controls the behavior of other composite Web service. We refer to such composite Web service as a supervision composite Web service.

For example, we show a supervision composite Web service for the document translation composite Web service in Fig. 4. This supervision composite Web service counts the number of invocations of the machine translation Web service and changes the endpoints to another machine translation Web service when needed.

Before invoking the document translation composite Web service, the supervision composite Web service is invoked. First the supervision composite Web service sets a breakpoint (`setBP`) before the invocation of the machine translation Web service `translate` in the document translation composite Web service. It also sets invocation of `count` in the supervision composite Web service as the callback Web service for the breakpoint.

Next, the supervision composite Web service stops at `count (receive)` in the figure waiting for an invocation from the document translation composite Web service. When the document translation composite Web service is invoked and stops before `translate` according to the breakpoint set by the supervision composite Web service, `count`

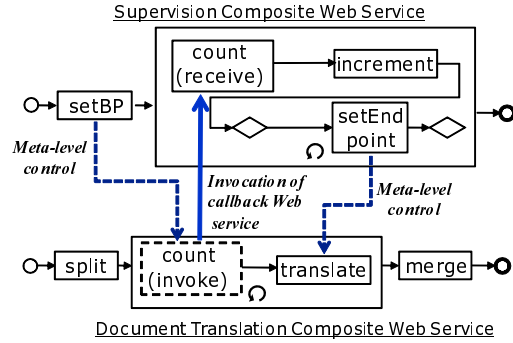


Figure 4. Supervision composite Web service.

is invoked as a callback Web service.

The execution of the supervision composite Web service is then resumed. The supervision composite Web service increments the recorded number of invocations of the machine translation service (`increment`). The number of invocations is recorded as a variable in the supervision composite Web service.

If the number of invocations of the machine translation Web service exceeds the limit, the endpoint of the machine translation Web service is changed (`setEndpoint`). Different from Fig. 3, `setEndpoint` in Fig. 4 is an invocation of one of the meta-level control functions, which is provided as a Web service.

The above steps are repeated according to the control construct for looping until all sentences of the given document have been translated.

`count (invoke)` at the document translation composite Web service is not a step included in the original definition of the document translation composite Web service. This is virtually realized by the meta-level function that sets a breakpoint and a callback Web service. Therefore, this invocation is represented by a rectangle with dashed line in Fig. 4. In the figure, a dashed arrow represents control of or setting of the state of execution by the meta-level control functions. A solid arrow is an invocation virtually set by the meta-level control functions.

The supervision composite Web service shown in Fig. 4 can be applied to various composite Web services that have some limit on the number of invocations of atomic Web service. The configurations that depend on the composite Web service to be controlled are locations of breakpoints and the location of Web service invocation whose endpoint should be changed. These configurations can be parameters of the supervision composite Web services. Therefore, we can specify the configurations before invoking the supervision composite Web service and do not have to embed the

Table 1. Meta-level control functions.

API	Effect
step	Execute the next activity in a composite Web service.
stop, resume, terminate	Stop/Resume/Terminate execution of a composite Web service.
getVariable, setVariable	Get/Set variable defined in a composite Web service.
setEndpoint	Set endpoints of an invocation in a composite Web service.
setExecutionPoint	Set the activities which is executed next.
setBP	Set a breakpoint at an activity in a composite Web service and a callback Web service invoked when the the execution stops at the breakpoint.

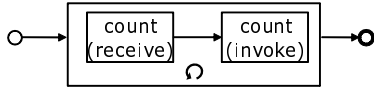


Figure 5. Choreography for definition of control protocol.

configurations.

This leads to the reuse of a supervision composite Web service as a general control pattern. Assuming that several supervision composite Web services are available, all that application developer has to do is to choose the appropriate pattern and set its parameters. This reduces the overhead in describing processes for coordination.

3.2 Choreography for Meta-level Control

Interactions between a supervision composite Web service and composite Web services to be controlled often needs to follow some protocols.

Assume that two instances of the document translation Web service try to invoke `count (invoke)` at almost the same time. The supervision composite Web service receives the request for `count (receive)` that arrives first and starts to increment the number of invocations. If the supervision composite Web service receives the request from another instance while incrementing the number of invocations, `count (invoke)` fails because the supervision composite Web service is not waiting for the request at `count (receive)`.

To solve this problem, we introduce choreography, which defines the protocol of interactions between a supervision composite Web service and the composite Web service being controlled. We adopt WS-CDL (Web Service Choreography Description Language), a standard language for choreography of Web services.

We show an example of choreography in Fig. 5, which defines the supervision composite Web service and the document translation composite Web service shown in Fig. 4.

Figure 5 also follows the notation of BPMN, but a rectangle which has a word inside represents an interaction be-

tween the supervision composite Web service and document translation composite Web service.

This protocol ensures that the execution of `count (receive)` in the supervision composite Web service and `count (invoke)` in the document translation composite Web service are processed in this order. This prevent one instance of the document translation composite Web service from invoking `count (invoke)` while the supervision composite Web service is executing `increment` at the request of `count` from another instance.

The choreography shown in Fig. 5 can be applied to any composite Web services that have some limit on the number of invocations with the supervision composite Web service shown in Fig. 4. Therefore we can reuse choreographies for typical controls.

The process for executing a supervision composite Web service and composite Web service to be controlled as defined by the given choreography is as follows.

After a supervision composite Web service and the composite Web service to be controlled are invoked, all interactions among the composite Web services and Web services invoked by them are monitored, and checked if the sequence of the interactions is accepted by the given choreography. If the choreography does not accept the sequence, we change the order of the interactions.

To realize the above, we implemented the architecture shown in Fig. 6. The architecture shown in Fig. 6 consists of two parts: Composite Web service execution engine and protocol control engine. When the composite Web service execution engine receives a request for invoking a composite Web service, the engine creates an instance of the requested service.

The supervision composite Web service and the composite Web service to be controlled interact with each other at breakpoints and callback Web services set by the supervision composite Web service via meta-level control functions.

The supervision coordinator in the protocol control engine monitors all the interactions defined in the given choreography. To process the interactions according to the choreography, the supervision coordinator performs the follow-

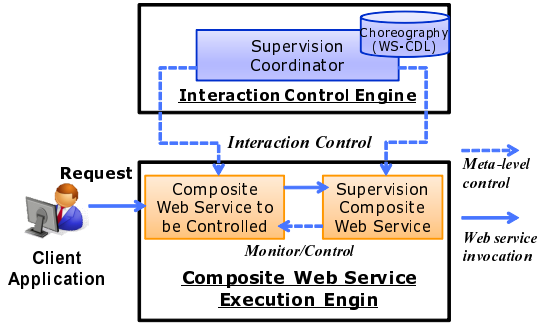


Figure 6. The implemented architecture.

ing steps when it observes that an interaction is going to occur: 1) Before the interaction is processed, put the information of the interaction into a queue which records the set of information of interactions. 2) Check the queue and process the interactions in the queue acceptable by the given choreography. 3) Remove the information of the interactions that have been processed from the queue.

If any sequence that consists of interactions in the queue is not accepted by the given choreography, execution of both the supervision composite Web service and the composite Web service to be controlled are halted and the control based on the choreography fails.

We extended one interpreter of WS-BPEL, ActiveBPEL⁴ to implement meta-level control functions on the composite Web service execution engine. We also modified one interpreter of WS-CDL, pi4soa⁵ to implement the protocol control engine.

4 Experiments

In this section, we show examples of applying Service Supervision in order to discuss its applicability and the performance. We then consider some of the features of Service Supervision.

We take the document translation composite Web service as an example and apply the following two controls.

- Assume the provider of the machine translation Web service limits the total number of invocations within some period. When the number of invocations exceeds the limit, change the endpoint of the machine translation Web service to select another machine translation Web service. (Control 1)
- Assume the provider of the machine translation Web service limits the number of concurrent accesses. When the number of concurrent accesses exceeds the

limit, change the endpoint to select another machine translation Web service. When the number of concurrent accesses falls under the limit, change the endpoint back to that of the initial service. (Control 2)

To implement Control 1, we used composite Web services shown in Fig. 4 and the choreography shown in Fig. 5.

To implement Control 2, we designed a supervision composite Web service that records the number of concurrent accesses to the machine translation service. This supervision composite Web service sets breakpoints before and after the invocation of the machine translation Web service in the document translation composite Web service. The callback Web services set for the breakpoints invoke the supervision composite Web service.

The supervision composite Web service waits for the invocation of the two kinds of callback Web services set before and after invocation of the machine translation Web service. It increments the recorded number of concurrent access at the first invocation, and then decrements the number at the second invocation. The choreography is similar to one shown in Fig. 5, but it selectively processes one of the two interactions that occur before or after invoking machine translation Web service.

First we inspected the execution time of the document translation composite Web service for each control in order to show that our method is practical. The input to the document translation composite Web service is a document consisting of 5 Japanese sentences to be translated into English. The limits placed on the total number of invocations and the concurrent access number limit are both 3.

Figure 7 shows the results. In this experiment, we compared the execution time of the control based on our framework, Service Supervision, with the execution time of the equivalent control realized by modifying the document translation composite Web service like Fig. 3. We executed multiple instances for each case. The time shown in the figure represents the average per instance of 10 trials.

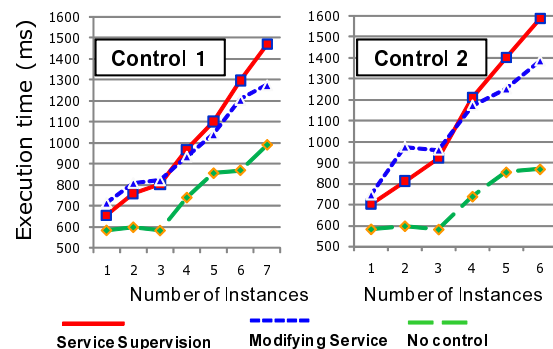


Figure 7. Comparison of execution time.

⁴<http://www.activevos.com/community-open-source.php>

⁵<http://pi4soa.sourceforge.net/>

The result shows that Service Supervision yields faster control than modifying the composite Web service in the case of 1–3 instances. This is because the invocation of `count` defined in the modified composite Web service takes longer than the access to information of other composite Web service via the meta-level control functions.

In the case of more instances, modifying the composite Web service yields faster control than Service Supervision. This is because the queue of the interactions which are not processed tends to have relatively many interactions and it takes time to check if the interactions can be processed or not by the WS-CDL interpreter.

The above result shows that Service Supervision proposed in this paper has disadvantage in scalability of number of instances compared to the existing framework. However, the decline of the performance is not serious in case that the number of active instances is small.

Next we compared the control complexity of supervision composite Web services to that of modified composite Web services in order to assess the cost of applying Service Supervision. We used the number of activities (atomic process step in a composite Web service including service invocation, copying variables, and interactions with other composite Web service) and containers of control constructs, and maximum depth of nested structures as metrics. To choose the metrics, we consulted a previous work on business processes that addressed such metrics[5].

Table 2 shows the complexity comparison results. For the supervision composite Web services, the sum of the numbers of activities and containers in the supervision composite Web services and the document translation composite Web service to be controlled in brackets because they are used together.

When using Service Supervision, the sum of the number of activities and containers exceeds that of the document translation composite Web service modified to realize the same controls. One reason for this is that the supervision composite Web services have activities for interacting with the client and the composite Web service to be controlled. Another reason is that some of the processes that require a control construct are separated into the supervision composite service and the composite Web service to be controlled. However, the difference is not significant because the difference in numbers of activities and containers does not increase with the complexity of the supervision composite Web service and composite Web services.

5 Related Works

Several researches have tried to change behaviors of a composite Web service without modifying the composite Web service. Most of the works have adopted the concept of AOP (Aspect-oriented Programming). Mosincat et

al. proposed a method for dynamic service selection and error recovery[10]. Their method transforms a BPEL process transparently to both the designer of the BPEL process and the execution engine. Moser et al. addressed that BPEL lacks functionalities for dynamic adaptation and monitoring[9]. They proposed a method for dynamic service selection and managing QoS information by intercepting SOAP messages. Dynamo[1] is also based on the concept of AOP. It monitors the messages exchanged between a BPEL process and external Web services and checks if the messages satisfy constraints.

The works mentioned above introduce their own languages or non-standard languages for defining the controls of composite Web services. However, there are already many systems which are working based on the current standard specifications. This may be a problem when we adopt the methods proposed in the previous works. Our framework proposed in this paper uses standard languages such as WS-BPEL and WS-CDL. Therefore we can exploit the existing tools like GUI editors and expertise of SOA engineers. On the other hand, our work requires modification of execution engine. This can be a disadvantage over the approach[10] to the execution engine.

AO4BPEL[4] also introduced AOP into BPEL. It allows the composite service designer to insert any process before/after an activity specified as a pointcut. Inserted processes are also described in BPEL.

This work allows us to define controls as BPEL processes. But it only adds a process into an existing BPEL process and cannot realize some controls such as stopping/restarting execution of a composite Web service which the meta-level control can perform.

Adaptive Workflow[12] focuses on workflows which are mainly executed by humans. It aims to adapt to unexpected exceptions and changes in the environment. Some previous works adopt case-based reasoning, rule-based systems, and planning to realize the adaptation [13, 3].

They focused on a method which realizes a feedback for execution of workflows by inspecting the result of components or modifying the workflows. On the other hand, our work provides a flexible runtime control framework for composite Web services. Therefore we can implement the previous works on our framework to solve a certain runtime problems.

6 Conclusion

In open environment, it is difficult for the designer of a composite Web service to know which atomic Web services are assigned for his composite Web service and how the services behave. This often causes runtime failure of the composite Web service due to the unique behavior of atomic Web services. However, the application developer who as-

Table 2. Complexity of composite Web services.

	# of activities	# of containers	max depth of nest
No control	9	2	2
Service Supervision (Control 1)	8 (17)	5 (7)	4
Modifying service (Control 1)	12	5	4
Service Supervision (Control 2)	10 (19)	6 (8)	5
Modifying service (Control 2)	15	7	5

signs atomic Web services and uses the composite Web service are generally not authorized to modify the composite Web service. Moreover the overhead of adding processes required for coordinating Web services can be too heavy for them.

This paper proposed Service Supervision, which monitors and controls execution of a composite Web service in order to coordinate Web services in the open environment.

The contributions of our work are as follows:

- We made it possible for application developers to change the behavior of a composite Web service without modifying the composite Web service by using meta-level control.
- We reduced the overhead of describing processes for coordinating elemental Web services in the open environment by allowing users to apply control patterns for coordination to various composite Web services.

The proposed architecture for Service Supervision is implemented on the existing frameworks such as WS-BPEL and WS-CDL. This allows us to exploit existing tools and expertise for designing or operating composite Web services.

In future work, we will extend our framework and coordinate composite Web services based on the policies of various stakeholders such as users and service providers.

Acknowledgments

This work was supported by Kyoto University Global COE Program: Informatics Education and Research Center for Knowledge-Circulating Society, Strategic Information and Communications R&D Promotion Programme from Ministry of Internal Affairs and Communications and a Grant-in-Aid for Scientific Research (A) (21240014, 2009-2011) from Japan Society for the Promotion of Science (JSPS).

References

[1] L. Baresi, S. Guinea, and P. Plebani. Policies and aspects for the supervision of BPEL processes. In *the 19th International*

Conference on Advanced Information Systems Engineering (CaiSE07), pages 340–354, 2007.

[2] A. Bramantoro, M. Tanaka, Y. Murakami, U. Schäfer, and T. Ishida. A hybrid integrated architecture for language service composition. In *IEEE International Conference on Web Services (ICWS-08)*, pages 345–352, 2008.

[3] F. Casati, S. Ilnicki, L. J. Jin, and V. Krishnamoorthy. Adaptive and dynamic service composition in eFlow. In *12th International Conference on Advanced Information Systems Engineering (CaiSE02)*, pages 13–31, 2000.

[4] A. Charfi and M. Mezini. AO4BPEL: An aspect-oriented extension to bpel. *World Wide Web*, 10(3):309–344, 2007.

[5] V. Gruhn and R. Laue. Complexity metrics for business process models. In *9th International Conference on Business Information Systems*, volume 85, pages 1–12, 2006.

[6] A. B. Hassine, S. Matsubara, and T. Ishida. A constraint-based approach to horizontal web service composition. In *the 5th International Semantic Web Conference (ISWC 2006)*, volume 4273, pages 130–143, 2006.

[7] T. Ishida. Language Grid: An infrastructure for intercultural collaboration. In *IEEE/IPSJ Symposium on Applications and the Internet (SAINT-06)*, pages 96–100, 2006.

[8] I. Matsumura, T. Ishida, Y. Murakami, and Y. Fujishiro. Situated web service: Context-aware approach to high speed web service communication. In *IEEE International Conference on Web Services (ICWS-06)*, pages 673–680, 2006.

[9] O. Moser, F. Rosenberg, and S. Dustdar. Non-intrusive monitoring and service adaptation for ws-bpel. In *17th International World Wide Web Conference (WWW 2008)*, pages 815–824, 2008.

[10] A. Mosincat and W. Binder. Transparent runtime adaptability for bpel processes. In *the 6th International Conference on Service Oriented Computing (ICSOC 08)*, pages 241–255, 2008.

[11] R. Tanaka, Y. Murakami, and T. Ishida. Context-based approach for pivot translation services. In *International Joint Conference on Artificial Intelligence (IJCAI-09)*, 2009.

[12] W. M. P. van der Aalst, T. Basten, H. M. W. Verbeek, P. A. C. Verkoulen, and M. Voorhoeve. Adaptive workflow on the interplay between flexibility and support. *the first International Conference on Enterprise Information Systems*, 2:353–360, 1999.

[13] B. Weber, W. Wild, and R. Breu. CBRFlow: Enabling adaptive workflow management through conversational case-based reasoning. In *7th European Conference on Advances in Case-Based Reasoning (ECCBR04)*, pages 434–448, 2004.