Programming Framework Based on Change-centric Web Service Evolution Model

Wei ZUO, Youssef Amghar, Aïcha-Nabila Benharkat

Université de Lyon, CNRS INSA-Lyon, LIRIS UMR5205,
20 Avenue Albert Einstein 69621, F-69621 Villeurbanne
Cedex, France
{wei.zuo, nabila.benharkat, youssef.amghar} @insa-lyon.fr

Abstract. A Web service always evolves during its lifecycle through continuously publishing new versions. Web service evolution is theoretically modeled in the community to help Web service stakeholders trigger and react to Web service evolution in a better way. From a technical perspective, the tasks in Web service evolution such as design, detection, execution, and adaptation to the Web service changes are undertaken by Web service developers. Unfortunately, few of the works, especially tools and methodologies, were specially taken to help the developers deal with dynamic evolutionary changes at programming level. In this article, we propose a framework based on our previous change-centric model to facilitate the developers to treat with Web service evolution. The framework supports the developers to execute, detect, and react to Web service changes at programming level.

Keywords: Web service; programming; evolution; adaptation

1 Introduction

A Web service evolves constantly during its lifecycle for two reasons: the consumers' changing requirements and the providers' improvements. As a result, the Web services are frequently changed by adding or updating new functions, new business processes, and new non-functional properties through publishing new versions. Especially in the systems which are built based on Service-Oriented Architecture (SOA), the Web services evolve even more quickly and frequently due to the large distributed scale and dynamic environment. In this case, it raises great pressure to treat with Web service evolution efficiently for all the stakeholders (consumers or clients, providers, developers, and brokers).

To face these challenges, the community tries to work out new models, tools, and frameworks to help the stakeholders manage, analyze, and adapt to the evolution of Web service. From a technical perspective, the tasks in Web service evolution such as design, detection, execution, and adaptation to the Web service changes are undertaken by the Web service developers who work at the programming level when dealing with Web service evolutions. Unfortunately, few of the works, especially tools and methodologies, were specially taken to help the developers deal with dynamic evolu-

tionary changes. In another word, the Web service developers need to be well equipped for updating and maintaining Web services in order to implement Web service evolution.

In this article, we focus on the development process in the evolution of the Web services and their client applications. We propose a new programming framework based on which it enables change-centric coding and client adaptation for Web service evolution. The main purpose of this programming framework is to reduce manual coding and reconfiguration when Web service evolves.

1.1 Change-centric Web Service evolution model

In our previous work [15], we have proposed a change-centric model for modeling Web service evolution at theoretical level. It defines 1) the roles and their behaviors during Web Service evolution, 2) the change specification which represents the changes between two versions, 3) the architecture of the system for change-centric Web Service evolution.

In change-centric model, the stakeholders in SOA concern different actions to cope with Web Service changes. The provider designs, applies the changes and them publishes the changes on the registry of broker. The consumer maintains the client applications of Web services and adapts to Web service evolution. The broker is responsible to maintain a registry of Web services and notify the consumers with the changes that they are interested in.

The change specification (CS) in change-centric model is built for formally describing Web Service changes. CS focuses on the changes of different aspects of Web Service which are included in Fig 1.

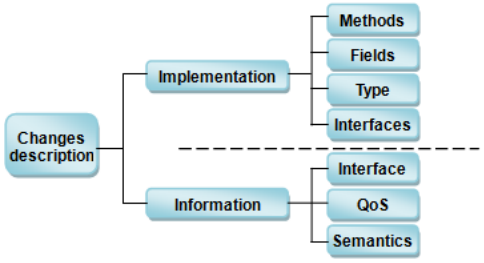

Fig. 1. Changes types of Web Service [15]

An instance of CS is so called delta in change-centric model. It is designed by provider, applied to a Web service by the developer, published on broker by provider, distributed by broker, and invoked or adapted by consumer.

Change-centric model is built at theoretical level to define the roles and behaviors in Web service evolution. However, two issues have not been taken into account: 1) what are the behaviors of the developers when the Web services evolve; 2) the implementation of programming methodologies has not been addressed.

1.2 Web Service adaptation

Adapting Web service stakeholders to the changes is one of the targets of research on Web service evolution. Proposals on Web service adaptation concern how to achieve backward-compatibility between the Web services and the client applications [6, 13]. Two types of adaptation are proposed: 1) adaptation at service side (or provider side) and 2) adaptation at client side. Adaptation at service side is limited by the usages of their consumers. So this article tries to follow the second type. When Web service evolves, the client applications are also required to take reactions to the evolution, or not they will fail in benefiting from the evolution. However, the developers who are responsible for the Web service client applications suffer from pain of manual adaptation when the Web services evolve frequently. And such manual adaptations are always invading into the modules that are related to the business. To achieve self-adaptation to the Web service evolution, they are also in short of support from the programming framework.

To deal with the issue, researchers are facing two challenges that are not well solved, including 1) to determine the compatibility between Web service and its client application and 2) to perform runtime and non-invasive adaptation dynamically.

Generally concluding the research situation, in this article, we present our programming framework based on change-centric Web service evolution model to provide a complete solution dealing with the technical problems in Web service evolution: 1) changes execution during Web service evolution at the service side; 2) client adaptation according to evolved Web services at the consumer side.

The following chapters will be organized as follows. In Chapter 2 we will introduce the related works on programming evolution of Web services. In Chapter 3 we will present our programming framework based on change-centric model. In Chapter 4 we give the implementation of our model. In Chapter 5 we conclude the contributions and bring out the future perspectives.

2 Related works

We address our work in the field of Web service development and evolution.

Treiber M [8, 9] develops a programming model for evolvable Web services. The main purpose of this model is to provide support for automatic adaptation through a dynamic modification framework for Web service based on Gensis. They apply self-adaptation by developers [9]. However, they do not have a complete evolution model to explain when and how to perform self-adaptation. Moreover, it lacks of an event propagation mechanism to publish and notify their modifications to Web Services.

Kaminski P [1] introduces their adapter chain solution to manage Web service evolutions and to ensure backward-compatibility. However, the proposed solution requires the service provider to develop adapter for each version of Web service, which is a hard task for the developers in the current fast evolved service-oriented systems.

Fokaefs M [13] presents a similar solution to our framework to implement client adaptation at runtime. The authors have categorized the web service changes into different types and gave the adaptation algorithm for incompatible changes. However,

they did not explain how to build the mutual perception between the Web service and the client to ensure adaptation, and they also did not explain the way to monitor the changes.

Generally speaking, previous works lack of a complete solution to explain: 1) why, when and how the Web service evolves; 2) how to build the evolution model with considering the cooperation and conversation among different stakeholders of Web service; 3) how to implement Web service evolution and client adaptation.

3 Programming model

Firstly, we define the behaviors of the Web service developer in Table 1 as a supplement to change-centric Web service evolution model.

Table 1. Behaviors of the Web service developers

Type of developers	Behaviors
Developer at service (provider) side	• Execute Web service changes • Publish the versions of Web service
Developer at client side	• Develop client applications • Adapt client applications to the Web service evolutions

Then, we introduce the architecture of the system based on change-centric model in Fig 2. At the service side, the provider performs the actions of design Web service changes. The developer at the service side executes and publishes versions of Web service. At the consumer side, the developer at the client side adapts the client applications to the Web service evolution. The steps of the client adaptation include the actions of subscription for new version events, impact analysis, and adaptation. At the broker side, the broker maintains a Web service registry and a database which stores the consumers' interests on the Web service evolution. The consumers' interests indicates the evolution events that the consumers concern.

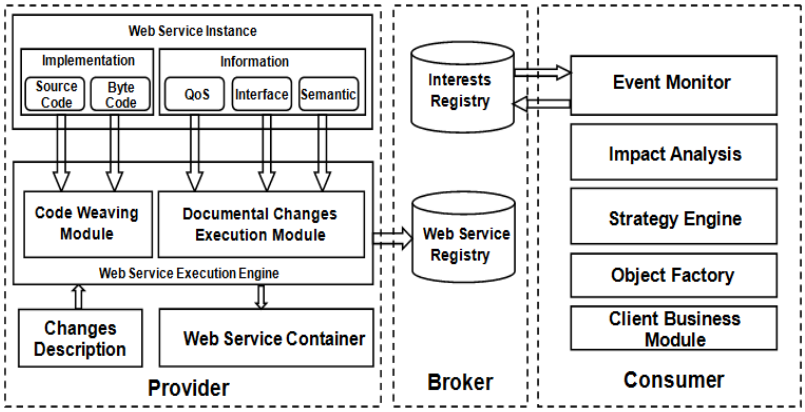

Fig. 2. System architecture for the programming framework in change-centric model [15]

The Changes Description at the provider side is a set of scripts that are designed for describing Web service changes, which follows the change specification of change-centric model. The Web Service Instance at the provider side is a set of primitives that are defined by change-centric model. The Web Service Execution Engine at the provider side executes the Web service changes that are described by Changes Description, generates new instances of Web services, and publishes Web services in both of the Web Service Container and the Web Service Registry at the broker side at runtime.

The Event Monitor at the consumer side subscribes and handles the events of Web service changes from the broker. The Impact Analysis at the consumer side analyzes the impact of the Web service changes to determine the compatibility between the client applications and the Web service. The Strategy Engine provides a set of strategies to adapt the client applications to the Web service changes. The Object Factory generates dynamically references to the Web services. The Client Business Module contains the client applications.

The programming framework for Web service evolution supports the two types of development in both Web service and the client sides. At the Web service side, Web service developer executes the changes to produce new versions of Web services. At the consumer side, the client application performs client adaptation to the Web service during Web service evolution.

Our contribution includes execution engine and adaptation engine at the service side, and the event monitor and adaptation engine at the client side. The other modules such as impact analysis will be treated in future.

3.1 Changes execution programming

In the change-centric model, if the provider wants to evolve a Web service, he always follows the steps of: 1) design changes, 2) apply or execute changes, and 3) deploy the new versions.

In the first step, the Web service provider specifies the targets and actions of the changes according to a model of a version of the Web service. We have defined the changes specification in an xml format which is used for network transferring and distribution.

For example, we have a change description of adding an operation as shown in Fig 3. It describes a change of adding an operation “bookTicketFromStarAlliance” to a existing Web Service in a new version 2.0.

And a change description of changing the XML Schema is shown as Fig 4. It describes a change of modification to the complex type Ticket.

```
<versionId=”2.0” previousVersion=”1.0” majorVersion=”1”/>
<change:interface type=”add”>
 <addSequence>
   <wsdl:operation name="bookTicketFromStarAlliance">
     <wsdl:input message="BookTicketInput"/>
     <wsdl:output message="BookTicketOutput"/>
   </wsdl:operation>
 </addSequence>
</change>
```

Fig. 3. Change specification of adding an operation

```
<operation action="modify">
<xs:complexType name="Ticket">
   <xs:Sequence>
     <xs:element name="date" type="xs:string"/>
     <xs:element name="flightNumber" type="xs:string">
     <xs:element name="seatNumber" type="xs:int">
   </xs:Sequence>
</xs:complexType>
<operation>
```

Fig. 4. XML schema change.

Fig 3 and Fig 4 show the changes description in change-centric model. They work as the requests of changing a Web service. For the developer at service side, they implement the changes by a set of scripting API as Fig 5 when they are informed by the requests. The API is designed for the Web service developers to modify one version of a Web service at runtime and publish the new version.

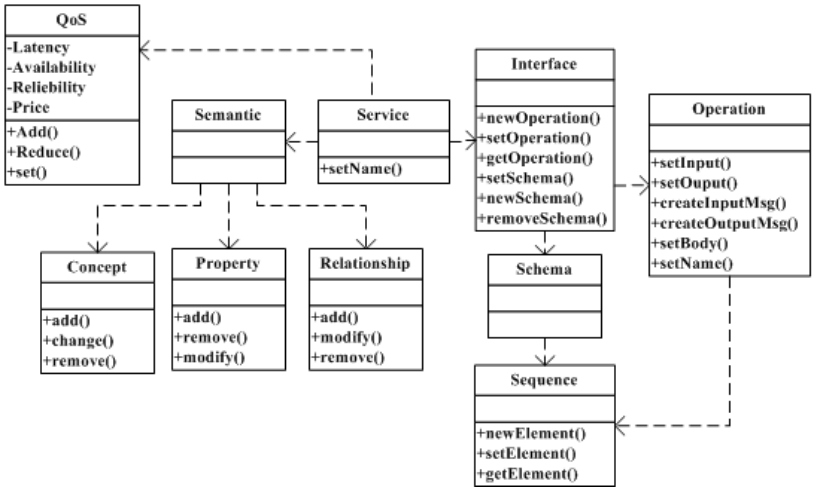

Fig. 5. Classes of service model in the framework

```
//get a copy of a service reference by specifying the director
y of the Web service with version 1.0.
Service s=copyof("/TransportService/V1.0");
//get the interface aspect reference
Interface i=s.getInterface();
//get the schema reference
Schema Sch=i.getSchemas().get(0);
//apply the changing actions of adding a new operation
Operation o=i.createNewOperation();
o.setName("bookTicketFromStarAlliance");
o.createInputMsg("BookTicketInput");
o.createOutputMsg("BookTicketOutput");
//modify the changing action of modifying a XML schema
Sequnce Seq= Sch.getComplexType("Ticket");
Seq.clear();
Seq.createElement("date" ,"xs:string");
Seq.createElement("flightNumber", "xs:string");
//publish a new version of a Web service
s.publish("http://localhost:8080/TransportService/V1.1");
```

Fig. 6. Example of Web service change description script

A part of the classes and operations that included in the API is shown in Fig 6.

The second step is to apply the changes and generate the new version of Web service. We provide an execution engine to analyze and execute the scripts described in the first step. The execution will be performed on both the description and implementation aspects of Web services based on Java platform and JAVASSIST framework developed by Shigeru Chiba.

The Web service instance is generated by the code weaving module of the execution engine. The input of the execution engine includes a complete Web service instance and the designed scripts from the first step. The output of the execution is a complete Web service instance identified with a new minor version including the documents, the source code and the byte code.

When a new version is generated by the execution engine, a change description will be also published on the broker. The broker maintains the registry of the Web services from different providers and stores the subscription information with the Web service consumers and the Web service evolution event that they are interested in. The data structure of the Web service registry in the Web service broker can be considered as a list of Web service descriptions. When a new version is published, the broker inserts a new element to the registry and posts an event to all the consumers who are interested in the new version.

3.2 Client adaptation

The programming framework at the service side generates the versions of a Web service and publishes the WSDLs or change descriptions into the registry of the Web service broker. To finish the evolution of the Web service, the client applications must

also take some actions to the Web service evolution. At the consumer side, the main behaviors of the client application include: 1) registering interests, 2) monitoring the new version event and analyze the impacts, 3) performing client adaptation.

The first behavior requires the system to build a conversation mode (Event Monitor) in a controlled way between the client and the broker. The second behavior requires the client to build extra modules (Impact Analysis, Strategy Engines) to deal with the changes descriptions which are notified by the broker. The third behavior requires another extra module (Object Factory) to support Web service invocation for the business modules of the client application. To sum up, an extra module at the client side similar as the adapters for adaptation at the service side is necessary. We call it client adaptation agent (CAA) as shown in Fig 7.

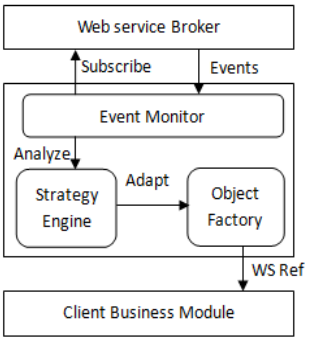

Fig. 7. Client adaptation.

CAA is mainly used for 1) producing Web service references for client application as an object factory, 2) communicating with Web service broker to subscribe interests and receive new version events, and 3) analyzing change impacts and perform client adaptation.

As shown in the Fig 7 above, Event Monitor is a communication module which subscribes and receives Web service evolution events. Strategy engine provides adaptation strategies according to each type of change in Web service. Object Factory produces the adapted Web service reference dynamically to ensure the Client Business Module works normally without manual adjustment or recoding.

The adaptation in the CAA generates a new proxy (also called stub or adapter) in Fig 8 at runtime which implements the interface to the old version of the Web service.

The strategy engine extracts adaptation suggestions from delta which are provided by the provider to assist the consumer with adaptation. The adaptation suggestions that can be provided by Web Service provider is shown in Table 2.

We emphasize that the adaptation is performed at runtime and is non-invasive into modules business of the client application. The adaptation process is totally transparent to the business modules. However, two things must be noticed: 1) it requires the client application to be organized based on our framework and Java platform and designed strictly in a principle of interface-implementation separation, 2) adaptation is only performed for temporary adjustment instead of manual operations which ensures

the normal functioning of the client application. The further work must be done if the client wants to be more compatible to the new versions of the Web services in its business process.

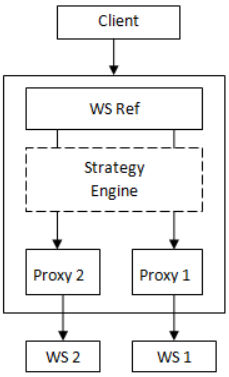

Fig. 8. Proxy generation

Table 2. Adaptation strategy at client side

Type of changes	Adaptation Strategy
Adding new elements to a complex type of Web service	Set default values for each elements
Deleting an operation from the Web service	Redirect the request to another optional Web service or set null.
Modifying the name of an operation	Redirect the request to the new operation
Modifying the XML schema	Modify the schema at the client side

4 Implementation

In this section, we introduce our initiative solution to implement the programming framework described in the previous sections.

4.1 Changes execution implementation

In section 3.1, the source code, byte code, QoS, interface, and semantics are under the control of Web execution engine. They are included in the Web service instance. The input of the execution includes the changes scripts designed by the developer and the version which need to be evolved. The output of the execution engine is another instance of Web service. The documental changes such as changes in WSDL or OWL-S are applied by a XML generator of deltas. We focus on the byte code execution.

In the programming framework, it is allowed to modify the fields, methods, parent, and interfaces of the Web service interfaces and implementation classes. Execution of changes follows the steps of:

1. Copy the WSDL of the specified major version and rename it to a new name space.

2. Modify the WSDL according to the change description and rename it into a new name space. (Depend on ow2-easywsdl).
3. Load the class or interface of the specified major version of the Web service and copy it to a new name <classname>+<versionId>.
4. Modify the class or interface according to the changes description by designer. (Depend on Javassist and only interface changes are available). If the change description includes changing an operation's body or adding an operation to a class, copy the necessary class.
5. Generate a new instance of the modified class and publish it to an address with a name identifying the new version. (Depend on apache-cxf and jax-ws).
6. Notify the broker with the changes and the address of the new version.

Notice that the generated service class is compiled and instantiate at runtime, the source code of the new version of the Web service is untouchable to the service developers, they are managed by the programming framework.

4.2 Client adaptation implementation

The client adaptation produces new class proxies which refer to the new versions of Web services.

1. Generate a new class which implements the current client stub of the Web service.
2. Generate a new interface to the new version of the Web service according to the changes.
3. Create a new member with the type of the new interface in step 2 for the new class in step 1 which refers to the new versions of Web service with correct format.
4. Set the body of the modified operation as calling the new operation.
5. Return a reference of a new instance of the generated class to consumer.

For example, assume that we have an interface as Fig 9:

```
package zw.provider;
@javax.jws.WebService
public interface ITransportService {
    public Ticket bookTicket(String date);
}
```

Fig. 9. Original interface class for TransportService

Now we want to rename the bookTicket operation to bookAirTicket. The generated new interface is as Fig 10:

```
package zw.provider;
@javax.jws.WebService
public interface ITransportService$v11 {
    public Ticket bookAirTicket(String date);
}
```

Fig. 10. Generated interface for the new version of TransportService

The generated new proxy class is shown as the following Fig 11.

```
package zw.provider;
import org.apache.cxf.frontend.ClientProxyFactoryBean;
public class ITransportService$v11$classProxy implements
ITransportService {
    public ITransportService$v11 ref;
    public Ticket bookTicket(String date)
    {
        return ref.bookAirTicket(String date);
    }
    public ITransportService$v11$classProxy(String serviceAddress)
    {
        ClientProxyFactoryBean factory = new
        ClientProxyFactoryBean();
        factory.setServiceClass(ITransportService$v11.class);
        factory.setAddress(serviceAddress);
        ref = (ITransportService$v11) factory.create();
    }
}
```

Fig. 11. Generated proxy class for the new version of TransportService

5 Conclusion

In this article, we introduce a programming framework which supports the Web service development at the service side and the adaptation at the client side. The features of the framework include:

Change-centric development: Evolving the Web services in a change-centric style can obviously reduce the side-effect caused by code duplication. It can also simplify the development process and make the developers focus on changes. The maintenance of changes instead of complete instances also reduces the pressure of version management.

Client self-adaptation: The programming framework supports the client application to adapt to the new versions of Web services even meet the incompatible changes. This can make the client application functioning continuously without shutdown or manual adjusting.

Transparency: The change execution engine is transparent to the Web service developer when modifying Web services. The client adaptation process is also transparent to the business modules of the client applications.

Non-invasive: The programming framework endows the Web service the ability of monitoring Web service at runtime without invasion into the development stage of Web service.

In future work, the most challenging topic is to ensure stateful evolution. Keeping the conversation between the client and the service uninterrupted during Web service evolution is crucial to the Web service evolution issue. It can reduce cost of Web service evolution for business and save time.

6 Reference

1. Kaminski P, Müller H, Litoiu M. A design for adaptive web service evolution[C]//Proceedings of the 2006 international workshop on Self-adaptation and self-managing systems. ACM, 2006: 86-92.
2. Wang S, Capretz M A M. A dependency impact analysis model for web services evolution[C]//Web Services, 2009. ICWS 2009. IEEE International Conference on. IEEE, 2009: 359-365.
3. Kajko-Mattsson M, Lewis G A, Smith D B. A framework for roles for development, evolution and maintenance of soa-based systems[C]//Systems Development in SOA Environments, 2007. SDSOA'07: ICSE Workshops 2007. International Workshop on. IEEE, 2007: 7-7.
4. Psaier H, Skopik F, Schall D, et al. A programming model for self-adaptive open enterprise systems[C]//Proceedings of the 5th International Workshop on Middleware for Service Oriented Computing. ACM, 2010: 27-32.
5. Feng Z, He K, Ma Y, et al. A Requirements-Driven and Aspect-Oriented Approach for Evolution of Web Services Composition[C]//Web Mining and Web-based Application, 2009. WMWA'09. Second Pacific-Asia Conference on. IEEE, 2009: 201-204.
6. Khater M, Malki M. An approach for adapting web services[C]//Multimedia Computing and Systems, 2009. ICMCS'09. International Conference on. IEEE, 2009: 56-61.
7. Na J, Gao Y, Zhang B, et al. Improved adaptation of Web service composition based on change impact probability[C]//Dependability (DEPEND), 2010 Third International Conference on. IEEE, 2010: 146-153.
8. Treiber M, Truong H L, Dustdar S. On analyzing evolutionary changes of web services[C]//Service-Oriented Computing–ICSOC 2008 Workshops. Springer Berlin Heidelberg, 2009: 284-297.
9. Treiber M, Juszczyk L, Schall D, et al. Programming evolvable web services[C]//Proceedings of the 2nd International Workshop on Principles of Engineering Service-Oriented Systems. ACM, 2010: 43-49.
10. Xie Q, Wu K, Xu J. QoS Driven Web Services Evolution[C]//Complex, Intelligent and Software Intensive Systems (CISIS), 2011 International Conference on. IEEE, 2011: 329-334.
11. Treiber M, Truong H L, Dustdar S. Semf-service evolution management framework[C]//Software Engineering and Advanced Applications, 2008. SEAA'08. 34th Euromicro Conference. IEEE, 2008: 329-336.
12. Fokaefs M, Stroulia E. WSDARWIN: A Decision-Support Tool for Web-Service Evolution[C]//Software Maintenance (ICSM), 2013 29th IEEE International Conference on. IEEE, 2013: 444-447.
13. Fokaefs M, Stroulia E. WSDarwin: automatic web service client adaptation[C]//CASCON. 2012: 176-191.
14. Banati H, Bedi P, Marwaha P. WSDL-temporal: An approach for change management in web services[C]//Uncertainty Reasoning and Knowledge Engineering (URKE), 2012 2nd International Conference on. IEEE, 2012: 44-49.
15. Wei Zuo; Benharkat, A.N.; Amghar, Y., "Holistic and Change-centric Model for Web Service Evolution," *Services (SERVICES), 2014 IEEE World Congress on* Services, vol., no., pp.250,253, June 27 2014-July 2 2014.