Holistic and Change-centric Model for Web Service Evolution

Wei Zuo, Aïcha Nabila Benharkat, Youssef Amghar
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*—Under the constantly evolving requirements from the consumers and competition pressure from the peers, Web Service providers are always striving to improve their services through publishing new versions. As more enterprises chose to embrace SOA, the frequent updates of Web services and increasing distributed environments have resulted in major challenges for all stakeholders to address the evolution of the Web service As a result, lots of solutions have been proposed to deal with the issues caused by Web Service evolution such as models, monitor, analysis, versioning, adaptation, and execution. However, few of them concentrate on the solution that covers all the evolution-related issue under one holistic model which explains 1) what has been changed, 2) when the changes occur, 3) how to apply changes, and 4) how to react to the changes. In this article, we present a change-centric model for Web Service evolution and explain how it deals with the evolution-related issues.**

Keywords—Web Service; evolution; changes; adaptation

I. INTRODUCTION

As more enterprises chose to adopt Web Service as their software framework to provide business services to the public, systems based on SOA are facing a more dynamic and distributed environment. Under the great pressure from evolving requirements of consumers and competition with the peers, the Web Service provider is always speeding up to publish new versions of Web Service which are improved with new functions, better Quality of Service (QoS), or fixing bugs. As a result, several issues become more urgent for different stakeholders due to the evolution of Web Service in SOA. Lack of an efficient model for Web Service evolution, 1) the manufacture process of Web Service becomes more time-consuming and is accompanied with lots of repeating manual configurations and development, and 2) the Web Service evolution produces lots of misunderstandings between the Web Service and their consumers so that it is quite difficult react to evolution correctly, especially at runtime. In this work, we present our change-centric and holistic model for Web Service evolution. It supports and facilitates the manufacture process of Web Service in evolution (design, execution, and publishing), the propagation process, and the client behaviors to evolution (monitor, analysis, and adaptation).

II. RELATED WORKS

To solve the problems caused by Web Service evolution, the researchers contribute for different aspects of them. For Web Service providers, whose tasks include designing, developing, publishing, and managing Web Service versions, researchers help them with versioning methodologies, programming frameworks, Web Service changes models. For Web Service consumers, who must be aware of and react to Web Service evolution, researchers propose monitoring methods, adaptation strategies, and impacts analysis. However, current research works are quite limited to gain a better understanding and solution to Web Service evolution.

A. Evolution programming

Researchers propose programming frameworks for evolvable Web Services. The main goal of this issue is to facilitate and encapsulate the development of Web Service. Martin Treiber in [13] and L. Juszczyk in [14] propose a framework to program evolvable services based on their Gensis framework on Java platform. The framework provides simple APIs for the developer to easily modify Web Service at runtime. It also supports self-adaptation through migrating or replicating services between hosts as well as changing structural interfaces. Gensis can generate Web Services instance from Web Service descriptions based Apache Velocity templates. A plug-in concept is introduced to endow extensibility to the system. The main limitation of Gensis is that it lacks of event propagation mechanism.

B. Changes extracting

Several works [1, 2] try to help the consumers to extract Web Service changes from some available documents such as WSDL through specific tools and algorithms. Zhi Le Zou et al in [3] try to extract the useful information from the release notes of Web Services. Fokaefs in [11] proposes a VTracker approach to resolve differences between two WSDL interfaces. The similarity of the works with the “analyzer” proposed is to extract changes related information through comparing the different versions of existing Web Service descriptions. This method can is helpful to obtain Web Service changes. However, the disadvantages are also obvious. 1) The correctness and completion are not guaranteed. 2) Resolving and comparing documents bring unnecessary performance decreasing.

C. Changes management

Web Service changes are related to evolution. Managing changes is to provide policies or tools for different stakeholders to help them formulate the correct plans to deal with changes. Treiber in [10] propose an interesting approach to identify the changes as well as their trigger sources which include the roles in SOA. They also provide an impact analysis for each of the proposed changes. This is useful for the stakeholders to have a holistic view of the Web Service evolution. However, it does now explain why, when, and how the changes occur. Furthermore, the impact analysis only points out what is the impact and who it will affect. No further analysis is proposed for the quantified result of impact and the corresponding reactions to it.

Bruno in [12] proposes a decentralized change management architecture. He considers designing a change dissemination mechanism that can be transferred across domains. Bruno's change 2.0 is enlightening with the change design technique. However, the execution of the changes has not been explained.

D. Service compatibility

To determine Web Service compatibility is important to Web Service evolution because it is the foundation for the consumers to take reactions. Defined by Meriem Belguidoum in [4] and Vasilios Andrikopoulos in [5], service compatibility can be separated into horizontal compatibility which indicates the interoperability between Web Service and its consumer, and vertical compatibility which indicates the possibility of replacement from one version to another version. Becker in [15] concludes the structural compatible Web Service changes. He also introduces their algorithm to explain how to examine the compatibilities.

E. Corrective and preventive evolution

As presented by Vasilios Andrikopoulos in [5], approaches of evolving Web Service can be distinguished by corrective ones which fix the mismatch changes during adaptation and preventive ones which forbid the incompatible changes occurring.

Vasilios's approach pursues the preventive manner. He proposes a contractually-bounded Web Service evolution theory which is based on a formal description of the compatibility guideline widely recognized and admitted. Preventive evolution, also called compatible evolution, requires the Web Service evolves in a limited way and always ensure the correctness of T-shaped (in both horizontal and vertical) evolution. The preventive approaches emphasize to drive the Web Service providers from different domains to evolve Web Service gracefully under certain rules. They can guarantee the evolution results do not break the interaction or replacement compatibility, ensure service stability and the whole process completely automatically. However, the limitations also seem unfixable. In the horizontal dimension, the approaches similar as contract based evolution break the loose-couple principle between provider and consumer in SOA design. As SOA develops, the provider and consumer will be weaker in the control of each other. It is impossible to build a certain contract to limit the evolution behavior for both consumer and provider. In the vertical dimension, the Web Service evolution in incompatible way is inevitable.

Against to preventive approaches, another option is corrective. The proposed model in this work falls in this manner. The corrective approaches usually perform adaptation at service side or client side. Kaminski.P in [6] proposes a design technique with adapter chain deployed at service side to obtain backward compatibility for the clients. Khater in [7] introduces another approach of adaptation which setups adapters at both service side and client side. He uses an infinite state machine to simulate and adapt dynamically the behaviors. Fokaefs in [9] introduces a client adaptation architecture (WSDarwin) and their differentiate method of comparing interfaces under a predefined delta model. The adaptation is performed by dynamically generation of the client stubs. The corrective approaches can liberate the Web Service from constraint evolution so they seem more convincing than preventive approaches. However, the disadvantages as described in [8] are obvious. There is no guarantee to ensure fully automatically adaptation. Most of the approaches need to reform the system architecture to support adaptation. Some of them even require the stakeholders to touch the implementation layer of applications.

III. CHANGE-CENTRIC MODEL

A. Web Service delta

Web Service delta is a set of changes from one version to its next version of the Web Service. Delta comes from the mind of the Web Service provider at design time and is executed by the Web Service developer. Once determined, delta will never change again and it keeps unique during the whole lifecycle of the Web Service. It is the only consensus for all the stakeholders to understand Web Service evolution. Most of the current works consider Web Service delta as the result of evolution so they propose different approaches to find and model it. We take into account of another aspect of delta that can be considered as the template of Web Service evolution. Once delta comes out of the mind of Web Service provider, it drives all the stakeholders to take corresponding actions that are related to evolution. That means, all the actions when evolution occurs in SOA including: 1) designing, executing, and publishing versions belonging to provider, 2) storing and distributing versions belonging to broker, and 3) monitoring, analyzing, and adapting to versions belonging to consumer, are all performed under the guide of delta. For provider, Web Service development can be totally replaced by delta design; for broker, managing of versions can be totally replaced by managing deltas; for consumer, reactions to evolution can be totally replaced by reactions to delta.

Delta holds all the differences that distinguish Web Service from one version to another version, which is called the functionality of delta in the logic dimension. We also consider functionality in time dimension. In time dimension, delta is able to represent a definite state of a Web Service during a certain time interval in its lifecycle if the previous version in the delta is determined. In another word, delta can be used for describing and invoking Web Service instead of different

descriptions such as WSDL, OWL-S, QoS and etc, of course, with the original description of the version indicated in delta.

Fig 1 indicates that delta actually runs through all the processes in SOA, which is so called “change-centric”.

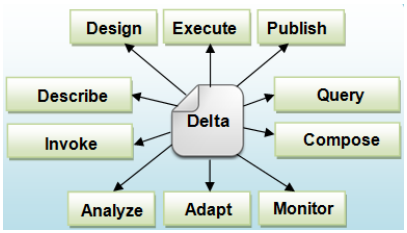

Fig. 1. Change-centric model

B. *Extended Web Service information model*

Current Web Service information model such as WSDL and OWL-S do not report on the version related information. To implement description with delta, the Web Service information model is extended with a Change Specification (CS). CS defines the changeable primitives with change target and change operators with a set of formal descriptions and standardized documental signatures. CS is designed by service provider and distributed to consumers and brokers. The changeable primitives in CS are shown in Fig 2.

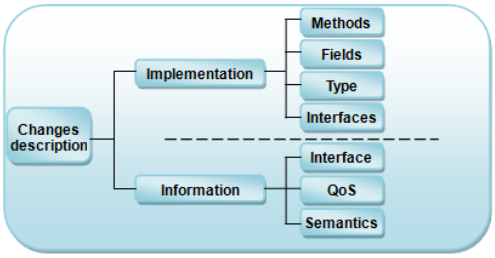

Fig. 2. Changeable primitives in CS

As explained above, delta is able to be used for Web Service description and invocation. Thereby a delta chain is proposed as Fig 3.

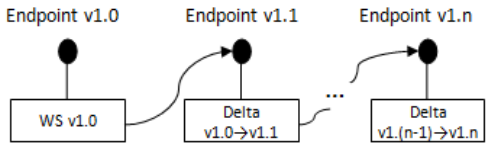

Fig. 3. Delta chain in Web Service

However, when the chain becomes longer, the cost of forwarding requests from a later version also increases. To avoid of developing a chain too heavy, there must be some breakpoints on the chain. These breakpoints are stable versions. For example, WS v1.0 is announced as a stable version which may be requested frequently. WS v1.1 is announced as a relatively unstable version. WS v1.0 reserves a complete instance of Web Service. WS v1.0 is deployed and functioning chronically. However, v1.1 only reserves the delta from v1.0 to v1.1after being published. When a request is forwarded to v1.1, the provider starts a process to generate the fully instance of v1.1. This generation is called as delta roll forward. Relatively, the generation roll back is also included in the model.

As lots of works mentioned, delta also functions as a result of evolution as well as a template. In the whole lifecycle of Web Service, some of the versions may be reserved and some of them may be retired. However for the deltas of a Web Service, they never change or reduce once being designed and are preserved forever unless the Web Service is definitively removed. So the accumulated delta of Web Services is the best option for historical analysis of the Web Service.

Another benefit of CS is that it also carries the other evolution related information such as adaptation assistant of each version from Web Service provider for the client adaptation just as [8]. As mentioned above, delta is the only consensus of all the evolution behaviors in SOA.

C. *System design*

Service-oriented architecture is typically described as a model with three roles (provider, broker, and consumer) and the interaction behaviors among them (lookup, bind, and publish). However, to cope with Web Service evolution, it lacks of 1) event propagation mechanism of evolution, 2) the behaviors that are related to evolution, and 3) the timing sequence of these behaviors. Fig 4 shows the extended SOA for evolution in the change-centric model.

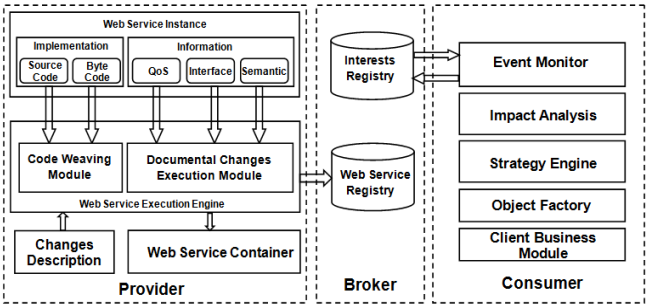

Fig. 4. Extended service-oriented architecture for Web Service evolution

Web Service designs the delta of Web Service, executes the delta into a new version with both implementation and information aspects of Web Service, deployed it to the Web Service container of a certain host, and notifies the Web Service registry of broker with the delta. The consumer and broker build a subscription/ notification conversation mechanism to propagate the delta over SOA.

D. *Programming Web Service*

In the service side, a programming framework must be provided to support Web Service development. The main task of the framework is to take a version of Web Service and a set of change actions as input and generate a new version (the delta). In this work, the Web Service execution engine is responsible to modify both the implementation and information aspect of Web Service. The implementation represents the underlying byte code which executes the business logic of the Web service. The information represents

the upper layer documents which describes different aspects of the Web service in text format (especially in XML) such as WSDL, OWL-S and so on. The output of proposed execution engine is CS which is well formed and distributed to the other stakeholders when evolving Web Service. The execution engine is also responsible for generating the full instance of a Web Service version when a request is forwarded to an unstable version.

E. Features

The proposed change-centric model proposes solutions which cover several aspects of Web Service evolution including model, architecture, and methods. The systems which are designed under this model can obtain numerous positive features.

a) Unambiguity: In this model, the designed delta plays the roles of both startpoint and result of Web Service evolution. There is no ambiguity in the system. The standardized CS and system design ensure that all the stakeholders have the same knowledge of delta. The system design also ensures that consumers can obtain the correct and complete CS. This feature is the basis to take the correct reactions to evolution for the other stakeholders.

b) Evolution without constraint: The proposed model adapt the services in a corrective way, which makes conditions for the Web to evolve without limitation though .

c) Programmable Web Service evolution: In the service side, this work proposed a set of tools and methods to generate Web Serices. A programming framework is provided for developers to modify Web Service at runtime. Compared to Gensis in [13, 14], the proposed framework allows the developers to directly modify the operation bodies and has a design to ensure event propagation.

d) Graceful versioning: Every version of the Web Service is accompanied with a delta. The creation of delta is under a well formed Change Specification, which makes the Web Service evolve in a more standardized way. A delta chain is proposed to manage versions of Web Service. It is used for simplify the versioning process and reduce the cost of infrastructures.

e) Dynamic adaptation: In the change-centric model, the client applications of consumer can monitor the delta event from broker and take adaption after impact analysis. An object factory is required in the client to produce the Web Service reference for business module. Similar as the other work [9], the client adaptation is also implemented by dynamic proxies generation and adaptation assitant from delta. By the way, the adaptation is also based on a consumer configuration.

IV. CONCLUSION

In this paper, we discussed about the background and the state of art in the field of Web Service evolution. Then we proposed a change-centric and holistic model to manage Web Service evolution. This, lead us to answer the questions that are proposed in the abstract.

1) What has been changed?: In change-centric model, we use Change Specification to describe formally the changeable primitives.

2) When the changes occur?: We explain that the changes occur at the design time of Web Service.

3) How to apply changes?: The change execution engine is our approach to apply changes.

4) How to react to changes?: We extend the SOA to obtain changes and perform client adaption to react to changes.

In future, there are still several challenges. For example, the change specification needs to be improved to support the other aspects of Web Service. There are some problems with the dependencies resolution when executing Web Service. Problems exist when evolving stateful Web Service.

REFERENCES

[1] Leitner P, Michlmayr A, Rosenberg F, et al. End-to-end versioning support for web services[C]//Services Computing, 2008. SCC'08. IEEE International Conference on. IEEE, 2008, 1: 59-66.

[2] Romano, Daniele, and Martin Pinzger. "Analyzing the Evolution of Web Services Using Fine-Grained Changes." Web Services (ICWS), 2012 IEEE 19th International Conference on. IEEE, 2012.

[3] Le Zou Z, Fang R, Liu L, et al. On synchronizing with web service evolution[C]//Web Services, 2008. ICWS'08. IEEE International Conference on. IEEE, 2008: 329-336.

[4] Belguidoum M, Dagnat F. Formalization of component substitutability[J]. Electronic Notes in Theoretical Computer Science, 2008, 215: 75-92.

[5] Andrikopoulos V. A theory and model for the evolution of software services[R]. Tilburg University, 2010.

[6] 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.

[7] 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.

[8] Andrikopoulos V, Benbernou S, Papazoglou M P. On the evolution of services[J]. Software Engineering, IEEE Transactions on, 2012, 38(3): 609-628.

[9] Fokaefs M, Stroulia E. WSDarwin: automatic web service client adaptation[C]//CASCON. 2012: 176-191.

[10] 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.

[11] Fokaefs M, Mikhaiel R, Tsantalis N, et al. An empirical study on web service evolution[C]//Web Services (ICWS), 2011 IEEE International Conference on. IEEE, 2011: 49-56.

[12] Wassermann B, Ludwig H, Laredo J, et al. Distributed cross-domain change management[C]//Web Services, 2009. ICWS 2009. IEEE International Conference on. IEEE, 2009: 59-66.

[13] 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.

[14] L. Juszczyk, H.-L. Truong, and S. Dustdar. Genesis – a framework for automatic generation and steering of testbeds of complexweb services. In ICECCS ’08: Proceedings of the 13th IEEE International Conference on on Engineering of Complex Computer Systems, pages 131–140,Washington, DC, USA, 2008. IEEE Computer Society.

[15] Becker K, Lopes A, Milojicic D, et al. Automatically determining compatibility of evolving services[C]//Web Services, 2008. ICWS'08. IEEE International Conference on. IEEE, 2008: 161-168.