DYNAMIC EVOLUTION OF SERVICE ARCHITECTURE IN MOBILE CLOUD APPLICATIONS

Huiqun Zhao[1], Jing Sun[1] and Xiaodong Liu[2]
[1]*Department of Computer Science, North China University of Technology, Beijing, China*
[2]*School of Computing, Edinburgh Napier University, Edinburgh, U.K.*

Keywords: Service-Oriented Architecture, SaaS in Cloud, Dynamic Evolution, Algebraic Model, Pervasive Systems.

Abstract: Although software services and service-oriented architecture have been researched widely, most existing research has focused on tools, process and methods of service engineering, and service semantics, and leave the formal specification of many aspects of SOA unsolved yet. One imminent challenge is the lack of the fundamental theoretic study of service architecture evolution, which is critical due to the very dynamic nature of services and service-based systems. This paper contributes to the state of arts by proposing a methodology that supports dynamic service evolution with respect to an algebra model of SOA. In this paper we infer and define service evolution as a paradigm of algebraic homomorphism mapping that can facilitate the analysis of the service architecture evolution with the algebraic method, e.g., the closure and the consistency of service evolution. In this way we develop two types of different service architecture evolution: one is for the reconfiguration of service composition inside a service system; and the other is for the collaboration composition between two service systems. The model has been applied to support the evolution of cloud-based services (as SaaS) which accept pervasive/mobile accesses. A case study combining with the transaction management and service adaptation is carried out in a context-aware tourism service application running in a multi-touch multi-user table providing "intelligent" offering of tourism information. Conclusions are drawn and future work is identified.

1 INTRODUCTION

Service Oriented Computing is an emerging paradigm for distributed computing and e-business processing. The Service Oriented Architecture and Web Service technique provide a systematic solution for building an application, for example WSDL, UDDI, SOAP and BEPL. However, current SOA technologies and standards are challenged by the constantly changing environment and user requirements. Concerns for runtime evolution are evoked to realize extensible and adaptive service-oriented applications.

Papazoglou (Papazoglou, 2008) observes the challenges of service evolution that the service technologies must be faced to. He classifies four type of changes in which the service may evolve: 1) *Structural changes,* 2) *Business protocol changes,* 3) *Policy induced changes, and* 4) *Operational behavior changes.* Mendling and Hafner (Mendling and Hafner, 2008) propose a novel definition called service choreography and the service orchestration for structural changes and protocol changes in comparison with paper (Papazoglou, 2008), and also contribute a method to dealing with their evolution. Andrikopoulos, Benbernou and Papazoglou (Andrikopoulos et al, 2008) argue that a service system can provide service evolution management, which provides an understanding of change impact, service changes control, tracking and auditing of service versions, and status accounting. Ryu et al (Ryu et al, 2008) declare that the business protocol evolution may raise critical problems: one of the most critical issues is how to handle instances running under the old protocol when it has been changed. Ping Yu, at el (Yu and Ma and Lu, 2005) introduces a new approach to service evolution. Taking advantage of experience from software architecture, especially dynamic software architecture (DSA) evolution management, they take service composition into account in runtime evolution and service evolution is referred to as dynamic reconfiguration of service architecture. The method has no formal basic and its viewpoint on service evolution is different from our algebra model based approach. Furthermore, many other

Zhao H., Sun J. and Liu X..
DYNAMIC EVOLUTION OF SERVICE ARCHITECTURE IN MOBILE CLOUD APPLICATIONS.
DOI: 10.5220/0003899102710274
In *Proceedings of the 2nd International Conference on Cloud Computing and Services Science* (CLOSER-2012), pages 271-274
ISBN: 978-989-8565-05-1

researchers pay much more attention to dynamic service selection and dynamic service composite which is related service evolution (Zeng and Benatallah et al, 2004) (Wan and Ulrich and Chen, 2008)(Zend and Sun, et al, 2010).

In this paper, facing the challenge of lacking the basic theoretic study of service architecture evolution, we propose a methodology, referred to as an algebraic model of SOA, which supports the evolution design of a dynamic SOA system. We then define and prove two algebra patterns that can guide service evolution. The proposed algebraic model has been applied to support the evolution of cloud-based services (as SaaS), which accept pervasive/mobile accesses from clients in multiple modal interactions. To demonstrate the capability of the algebra patterns, we use the *Apto* tool to carry out a case study, which involves the transaction management and service adaptation in a context-aware tourism service application.

The reminder of the paper is organized as follows: Section 2 introduces the algebraic model of SOA. Section 3 defines the dynamic evolution of SOA based on its algebraic model. Section 4 presents the case study of pervasive service evolution in the cloud. Finally, section 5 presents the conclusion and future work.

2 AN ALGEBRAIC MODEL OF SOA

In this section, we define an algebraic model of SOA based software application. The model provides a foundation for the software design and the further evolution of the SOA based application. We formally define the concepts and verified facts of SOA in the algebraic model, such as service component definition, service composition as an operation and its attributes, the descriptive definition of SOA.

Definition 2.1 (Service Component). A service software component (or component for short) comes from several different forms, including a module of program code, a programming language, a computational unit under the execution environment of the language, a data item, or an item of the data processing result. A component consists of an interface and an implementation module.

The interface of a service component is a set P of external ports, P={Porti, i=1,…,n}. Each port Porti is an 8-tuple: <ID; Pbuli; Privi; Extni; Behai; Megsi; Consi; NonFunci>, where

ID is the identifier of the component or a code for interpreting its IP address.

$Publ_i$ is the set of functions (or activities) the port provides to the external environment and other components. The execution of an activity is noted as x.

$Extn_i$ is the set of functions (or activities) of the external environment and other components that the port interacts with.

$Priv_i$ is the set of functions (or activities) used only internally to this port.

$Beha_i$ is a description of the semantics of the port's behavior, consisting of a set of predicate logic expressions.

$Megs_i$ is the set of messages generated at this port, expressed as event signals.

$Cons_i$ is the set of constraints to the port consisting of the initial, pre-and post-conditions for the execution of the component, and may be expressed as Cons(init; pre-cond; post-cond). In general, we can assume that the initial condition is always satisfied. In these cases, init is omitted in the expression.

$NonFunc_i$ is a description of non-functional aspects of the port, including descriptions for the service strategy, contract, security, and reliability.

In the SOA software algebraic model, a set of operations of service composition are defined, including "trigger", "use", "collaboration", "parallel", "repeat", and "selection". Among them, the operation Trigger depicts the behavior of the message-based component composition, and is represented as $A \oplus B$ or $x \oplus y$ where x and y are the activities of component A and B respectively.

The SOA algebraic model clarifies the attributes of SOA as an algebraic system, and provides the formal algebraic model definition of SOA. We call SOA=<C,O> the algebraic model of SOA, also the algebraic expression of SOA. Here, C and O represent the set of service components and the set of the operations of service composition. We here prove that SOA=<C,O> constructs an algebraic system, that is:

Theorem 2.1. Let SOA = <C, O>, the SOA forms an algebraic system to each composition operation in O.

3 DYNAMIC EVOLUTION OF SOA STRUCTURE

First we give the definition of the evolution of a service component.

Definition 3.1. Let A and B be two service components in Dom(U). B is said to be an evolution from A, denoted Evolve(B;A), if the following conditions are met:

Dom(B) = Dom(A) (2-2a)
Publ(B) ⊇ Publ(A) (2-2b)
Extn(B) ⊆ Extn(A) (2-2c)
Priv(B) ⊆ Priv(A) (2-2d)
Beha(B) ⇒ Beha(A) (2-2e)
Msgs(B) = Msgs(A) (2-2f)
Cons(B) =Cons(B) or Cons(B)⇒Cons(B) (2-2g)
NonFunc(B)=NonFunc(A)
or NonFunc(B)⇒NonFunc(A) (2-2h)

The evolution of service component does not affect the organization of services, therefore has no direct impact on the architecture, i.e., no change on the organization of the SOA system. Below we define the evolution of the architecture and show its impact on the organization of services in SOA applications.

Definition 3.2. Let $S_1=<C_1,O_1>$ and $S_2=<C_2,O_2>$ be two different services. If $\forall x\in C_1$ always holds $\exists y\in C_2$ and y is a mapping to x, then we assert that there is a mapping between S_1 and S_2, denoted $f:S_1\rightarrow S_2$, or y=f(x). If the mapping is full mapping, then the mapping between the services is full mapping; if the mapping is one to one, the mapping between the services is also one to one mapping.

Based on definition 3.2, the definition of SOA architecture evolution is further refined as follows:

Definition 3.3. Let $S_1=<C_1,O_1>$ and $S_2=<C_2,O_2>$ be two different services. F is a mapping from S_1 to S_2. If $\forall x\in C_1$ and $\forall y\in C_1$ such that $f(x\otimes y) = f(x) \oplus_j f(y)$, where $\otimes\in O_1$ and $\oplus\in O_2$, then we say the f is **homomorphic evolution** from S_1 to S_2. S_1 and S_2 are also said to be **homomorphic architecture**. If f is a one-way mapping, then f is said to be a one-way **homomorphic evolution** from S_1 to S_2. If f is a one-to-one mapping, then f is said to be an **isomorphism evolution** from S_1 to S_2, and S_1 and S_2 are **isomorphic architecture**.

The evolution of service system may be caused by various reasons: i) service system evolution which is caused by the evolution of service components, which is a passive evolution to the service developer; ii) service system evolution which is caused by the evolution of the system architecture, which is considered as an active evolution. Active evolution means the improvement of the quality of services and the original service system. Here we present one type of the architecture evolution, which only involves the adjustment of the mode of service composition (operation).

Theorem 3.1: For any SOA=<C,O>, it is always possible to evolve into a composition of service components which only involves the composition of invocation operations Op^ and selection operations +.

Theorem 3.2: Let $F=a_1f_1+\ldots+a_kf_k$ and $G=b_1g_1+\ldots+b_kg_k$ be two service expressions, then an isomorphism evolution must definitely exist based the collaboration operation Θ, which makes the new service have the isomorphism architecture of FΘG.

4 CASE STUDY

As a proof-of-concept we implemented a prototype tool (*Apto*) for the service evolution with process variant generation technology (Jaroucheh et al, 2010). We present here a case study for a tourism service application running in a multi-touch multi-user table that provides "intelligent" offering of tourism information. The service is deployed as a mobile SaaS in the cloud "E-Napier", which consists of 40 PCs in grid managed by VMWare. The aim of this application is to provide users with life experience when booking for their holidays. Obviously, since usually different users use this type of applications simultaneously, considering and resolving conflicts between users' preferences (part of the application context) become important. This application runs in a travel agency which has some agreements with other travel tourist agents distributed in different cities. These agents provide Web service interface for others to get offers and book for their trip. To give users a life experience the application displays customized information about each city i.e. historic, artistic, etc. according to the users' preferences. In addition, it tries to find their friends who are actually located (or have been) in these cities.

4.1 Specification of the Original System

The original service system architecture is described in Figure 1. It employs a transactional mode to serves each user within one event cycle. It is a big challenge for a decentralized service system to maintain atomic event cycle especially when the user number increasing dramatically.

The algebraic model of original service system present as 4-1.

$$\begin{cases} Service1\Theta(Agent1+Agent2+Agent3) \\ Servcie2\Theta(Agent2+Agent3) \end{cases} \quad (4\text{-}1)$$

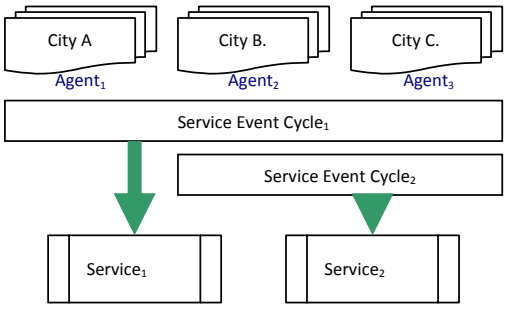

Figure 1: Original service model.

4.2 Architecture Adaptation

To satisfy the increasing business requirement a substituted service mode has been adopted which guided by the Theorem 3.2. For keeping the consistence with our service model theory, the formal description of the two service modes is given as follow.

$$\begin{cases} Service4 \otimes \\ (Servcie1 \Theta gent1 + Servcie2 \Theta Agent2 + Servcie3 \Theta Agent3) \\ \quad Servcie5 \otimes (Servcie2 \Theta Agent2 + Servcie3 \Theta Agent3) \end{cases} \quad (4\text{-}2)$$

Meanwhile the formula 4-2 lets Service1 collaborates with Agent1-3 individually in according with the Theorem 3.2. In order to gather the service information from the parallel Agent1-3, a service composition, namely service trigger, is executed.

We have designed and implemented a self-adaptation approach to the service evolution. When the user number is below a valve we still use the original service mode, and if the traffic peak appears, we then change to the new service mode.

5 CONCLUSIONS

The objectives of the research are to develop an algebraic model for service evolution at both architecture and service levels and then to identify and formally define various service evolution patterns. Our literature review has shown that the proposed algebraic model has novel contributions and similar work has not been done.

Based on the algebraic model of SOA, the paper discusses the methodology of the dynamic evolution of service architecture, proposes that service mapping can be viewed as a basis of the evolution of service architecture, and consequently views evolution as a process of system transformation instead of simply focusing on local adjustment. The paper proposes two fundamental methods for the dynamic evolution of service architecture, and demonstrates the feasibility of the methodology through both theoretical proof and a practical case study of cloud-based pervasive service evolution.

In the future we will explore the relationships between the evolution of service components and service architecture, and the impact of the cloud features such as dynamic agility and virtualisation on the architectural evolution of complex cloud-hosted SOA systems.

ACKNOWLEDGEMENTS

The work in this paper has been jointly sponsored by the British Royal Society of Edinburgh (RSE-Napier E4161) and the Natural Science Foundation of China (Ref: 61070030).

REFERENCES

Andrikopoulos, V., Benbernou, S. and Papazoglou, M. P., 2008. Managing the Evolution of Service Specifications. *CAiSE 2008*: pp. 359-374.

Jaroucheh, Z., Liu, X. and Smith, S. 2010. A MDD-based Generic Framework for Context-aware Deeply Adaptive Service-based Processes. *The 8th IEEE International Conference on Web Services (ICWS'10),* Miami, USA.

Mendling, J. and Hafner, M., 2008. From WS-CDL choreography to BPEL process orchestration, *Journal of Enterprise Information Management*, 21(5).

Papazoglou, M.P., 2008. The challenges of Service Evolution. *Advanced Information Systems Engineering. LNCS*, Vol. 5074/2008, pp. 1-15.

Ryu, S.H., Casati, F., Skogsrud, H., Benatallah, B. and Régis S.P., 2008. Supporting the Dynamic Evolution of Web Service Protocols in Service- Oriented Architectures, *ACM Transactions on the Web*, 2(2).

Wan, C., Ullrich, C., Chen, L. et al., 2008. On solving QoS aware service selection problem with service composition. *In Proc of the 7th International Conference on Grid and Cooperative Computing*. Shenzhen: IEEE, pp.467- 474.

Yu, P., Ma, X. and Lu, J., 2005. Dynamic Software Architecture Oriented Service Composition and Evolution. *Fifth International Conference on Computer and Information Technology (CIT'05)*. Shanghai, China.

Zeng, L. Z., Benatallah, B., Ngu, A. H., Dumas M., Kalagnanam, J., and Chang, H., 2004. QoS-aware middleware for Web services composition". *IEEE Trans. on Software Engineering*, 30(5), pp. 311−327.

Zeng, J., Sun, H., Liu, X., Deng, T., and Huai J., 2010. Dynamic Evolution Mechanism for Trustworthy Software Based on Service Composition, *Journal of Software*, 21(2), pp.261−276.