Hindawi
Scientific Programming
Volume 2017, Article ID 5725864, 11 pages
https://doi.org/10.1155/2017/5725864

Research Article

A Clustering Method for Isomorphic Evolution of Web Services

Qiang Hu, Zhen Zhao, and JunWei Du

School of Information Science and Technology, Qingdao University of Science and Technology, Qingdao 266061, China

Correspondence should be addressed to JunWei Du; djwqd@163.com

Received 6 April 2017; Accepted 5 July 2017; Published 7 September 2017

Academic Editor: Fabio La Foresta

Since the basic binding unit of current service request and response schema is an atomic Web service, it needs to costly find a substitute service or reconstruct the service process in the original service space once a fine granular evolution requirement occurs. To reduce the complexity of fine granular service evolution, an isomorphic evolution mechanism based on service clusters is proposed. Searching space can be reduced and responding flexibility will also be improved by adopting service cluster as the unit of service response. Simple evolution and merging evolution were put forward to handle the evolution of atomic Web services. Meanwhile, a formal model and the quality computing method for service processes built by service clusters were presented based on the logic Petri nets. Two types of evolution patterns including dot isomorphic evolution and chain isomorphic evolution were proposed to evolve service processes. The algorithms for different isomorphic evolution patterns of atomic service and service processes were designed in the paper. Simulation experiments were conducted on 10000 Web services with different process patterns. Compared with the traditional service request and response schema, the efficiencies of service discovery and isomorphic evolution are improved greatly in our proposed method.

1. Introduction

With the maturity of the new information technologies, such as cloud computing, mobile Internet, and the Internet of Things, the developing method and deploying pattern in the network software systems outline the remarkable service-oriented features [1]. Service-oriented architecture (SOA) is frequently adopted to design and develop the network software while more business functions are encapsulated and published as network information services in Internet by the enterprises or organizations [2]. Among different technologies to build network information services, Web service is the most popular way to develop the network software system based SOA. Currently, all of the major software developing tools can provide the support to design and deploy Web services. As the remote-called procedure modules, Web services were often designed with smaller granularity. Thus, a group of Web services were often assembled as a service process to cater for the complicated business requirement which cannot respond by the single Web service [3].

Whether it is the atomic Web service or service process, they are all working in the fast-changing distributed environment. Some services may be invalid during their lifecycle. Meanwhile, slight changes of users' function or quality requirement may also occur during the execution of the composite service process. Thus, an evolution mechanism needs establishing to make the service-oriented software automatically adapt the variations of external application environment and interior users' requirement.

The main work of service evolution is to find a new service (or service process) catering for the changed application environment or service requirement and replace the invalid one. A series of research achievements have been obtained for different evolution requirements and scenarios. Zuo et al. introduced the change-centric model which describes the behaviors of the stakeholders in Web service evolution [4]. Mechanisms for recommending Web services evolving in heterogeneous environments were widely concerned. A new agent-based system was proposed to recommend multimedia Web services based on the effect of the currently exploited device in [5]. Underlying search and ranking algorithms that enable service recommendations was studied in [6], and service recommender system using enhanced syntactical matching was proposed. A hybrid method which combines

collaborative filtering and content-based recommendation is presented in [7]; it can dynamically recommend Web services which fit users' interests. A recommendation visualization technique that employs the characteristic of QoS and achieves considerable improvement on the recommendation accuracy was proposed in [8]. Web service candidates computed the score by measuring their relevance with historical and potential user interests and their QoS utility. A diversity-aware Web service ranking algorithm was designed to recommend the Web service candidates based on their scores [9].

A trust-based approach for large-scale federations utility computing infrastructures was proposed in [10]; it can find the most suitable collaborators avoiding exploration of the whole service space. Furthermore, a multiagent protocol for service level agreement negotiation was proposed in [11]; it can give a helpful enlightenment for service locating and substitution in the service evolution. An agent-based cloud service composition approach was presented in [12] and the self-organizing agents made use of acquaintance networks and the contract net protocol to evolve and adapt cloud service compositions. Similarly, an evolutionary adaptation mechanism was designed using genetic algorithms for agents to evolve their behaviors and improve their fitness values to the environment [13]. Wang et al. proposed a service evolution model to analyze service dependencies, identify changes on services, and estimate impact on the users [14]. A framework of service evolution and reuse was designed by Schumm et al. [15]. Service processes can be customized or reused by the operations provided, such as publishing, retrieval, extraction, and integration, in the proposed framework and the sharing repository of service processes. Mohammad et al. proposed an evolution method of SasS services based on the biological heredity and illuminated an effective way to customize service process from the perspective of heredity in [16].

The existing research about service evolution can be summarized as the following aspects: service version management [17], service migration [18, 19], service substitution [20, 21], and structure evolution of service processes [22]. These works can also be divided into two categories: heterogeneous service evolution and isomorphic service evolution. Decomposition, truncation, link, and synthesis are the basic transformation operations in the heterogeneous service evolution. It means that the structure of a service process was significantly changed; that is, the process logic and component services were vastly varied. Thus, heterogeneous service evolution is very time-consuming and often achieved by static customization [23].

Conversely, few Web services or very short piece of a service process was adjusted in the isomorphic service evolution. Isomorphic service evolution is widely utilized to address the slight variations of service requirement or working conditions. However, the current research of isomorphic service evolution mainly focused on how to substitute the invalid service with the service context. Few studies have been investigated on isomorphic evolution of service processes [24].

To simply address the isomorphic service evolution, we propose a service cluster-based evolution mechanism. The main contribution of this work is as follows: (1) A three-tiered service request and response architecture oriented service clusters are put forward. (2) An isomorphic evolution mechanism for service processes is proposed in the above architecture. (3) A method to compute service quality in different process patterns is also given based on the logic Petri net.

Locating the alternative service is the vital work in process of service substitution or process reengineering. Similar Web services were mapped as a service cluster in our three-tiered service request and response architecture. Once a Web service needs replacing, the alternative one can be searched in its subordinate service cluster. Search space for the alternative service is restricted within one service cluster and it is greatly reduced. Meanwhile, two types of evolution algorithms were provided in the isomorphic evolution mechanism. They can replace invalid services at the level of service clusters and rapidly reconstruct service process. Compared with the original one, the newly generated service process can support the same business function and (even better) service quality. Thus, service substitution or reconstitution for service processes can be easily achieved by the proposed isomorphic evolution method.

The rest of the paper is organized as follows. Section 2 introduces the service cluster oriented isomorphic service evolution mechanism. Isomorphic evolution algorithm for atomic Web service is presented in Section 3. Two patterns of isomorphic evolution for service processes and their corresponding evolution algorithms are elaborated in Section 4. Section 5 presents the simulation experiment. And conclusions are given in Section 6.

2. Isomorphic Service Evolution Mechanism Based on Service Cluster

Service request generally responds by an atomic Web service in the traditional service-oriented architecture (SOA). Service composition is proposed to combine different Web services together to cater for complex service requirements [25]. Since the component units are atomic Web services, the substituted service needs searching in the existing service register space once some service is invalid in the composed service process. Seeking substituted service is a huge workload for the massive volume of similar Web services.

To reduce the difficulty of service substitution and conveniently achieve isomorphic service evolution for service processes, we need to increase the granularity of service responding units. Web services with similar function and service quality are logically encapsulated as a service group in this study, namely, service cluster. Service clusters are employed to respond to service request. Then service substitution and isomorphic evolution are performed based on the service clusters. Thus, service searching space can be greatly reduced by this group mechanism.

The traditional SOA is converted to a three-tiered service request and response architecture based on the service clusters by introducing the group mechanism. As shown in Figure 1, service clusters are located at a virtual resource layer.

They are the mapping collections of Web services with similar functions in the physical resource layer.

Service request is described in a business model layer and it can be disintegrated and can respond by a single service or service composition. The request of single service in business model layer responds by a service cluster while the request of services composition is undertaken by the combination of a group of service clusters. For any responding service cluster in the virtual resource layer, it will be ultimately bound by a specific Web service in the physical resource layer.

Service clusters are the service responding units in our proposed three-tiered service request and response architecture. Since service cluster is a group of Web services with similar functions, the substituted service can be easily obtained in the responding service cluster once the current Web service cannot work properly. The searching space of the substituted Web service is greatly reduced for the lookup limited in a service cluster. Meanwhile, the service process composed of the service clusters may evolve many similar process instances so it can meet more different service requirements. For the complex service request responding by a service process, the structure of service process in the virtual resource layer can maintain invariably when the service request or external environment is slightly changed. Thus, the isomorphic service evolution can be easily achieved based on service clusters.

Usually, one or more Web services can satisfy the service request in a service cluster and all these services are built as a set in our study, namely, the responding service collection. The responding service collection is symbolically denoted as $S_{\_rw}$. The substituted one can be selected from $S_{\_rw}$ when the atomic Web service is achieving the isomorphic service evolution. Similarly, for service processes, the isomorphic service evolution is divided into two categories: dot evolution and chain evolution. The isomorphic evolution for service processes can be conveniently implemented by the proposed evolution algorithm in the following section; for example, service process 1 can evolve into service process 2 in Figure 1 by replacing some services in their component service clusters.

Some concepts, such as service pool [26], service cluster [27], and service class [28], were proposed to model the collection of Web services. Web services in these concepts were required with the same function and interfaces. Thus, they are with small granularity of function and poor ability of isomorphic service evolution. Different from the above concepts, we do not restrict Web services included in the service cluster of our study with the identical function and interfaces. The formal definitions of Web service and service cluster are presented as follows.

Definition 1 (Web service). A Web service is a 6-tuple $Ws = (W_{\mathrm{Id}}, W_{\mathrm{des}}, I, O, Q, L)$, where

(1) W_{Id} is an identification of a Web service;

(2) W_{des} is a function description of the Web service;

(3) I and O are the sets of input and output parameters;

(4) Q is a set of quality parameters, where $Q = \{q_i\}$, $q_i = \{N, C, V, U\}$, N is the name of quality parameter, C is a comparison operator, V is the value of the parameter, and U is the unit of quality parameter;

(5) L is the URI of a Web service.

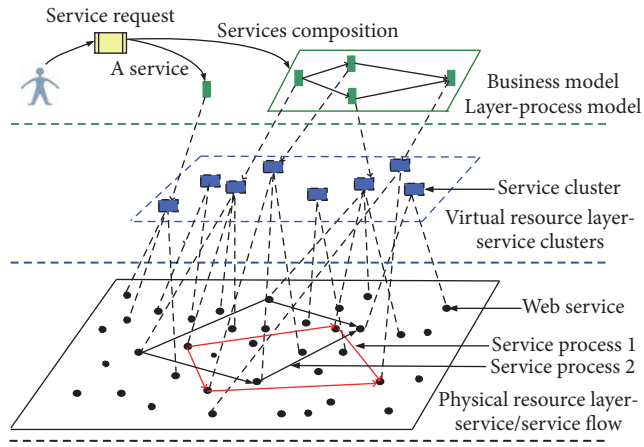

Figure 1: A three-tiered service request and response architecture oriented service clusters.

Definition 2 (service cluster). A service group is a 7-tuple $S_{\mathrm{cluster}} = (C_{\mathrm{Id}}, C_{\mathrm{des}}, I, O, S_w, Q_c, f)$, where

(1) C_{Id} is an identification of a service cluster;

(2) C_{des} is a function description of a service cluster;

(3) $I = Ic \cup Ip$ and $O = Oc \cup Op$, where Ic and Oc represent the common input and output parameters of a service cluster and Ip and Op represent the private input parameters and output parameters of some component services in the service cluster, respectively;

(4) S_w is the set of component services in a service cluster, $S_w = \{ws_1, ws_2, \ldots, ws_n\}$, where ws_i is a Web service and $1 <= i <= n$;

(5) $f: Op \rightarrow 2^{Ip}$, where $\forall o \in Op$, $f(o) = \{\langle ws.I \cap Ip\rangle \mid \forall ws \in S_w \wedge o \in ws.O\}$, and f is a function to describe the mapping relations between Op and Ip;

(6) $Q_c = \{q_i\}$, where Q_c is the set of quality parameters and $q_i = \{N, C, [V_{\min}, V_{\max}], U\}$, where $V_{\min}$ and $V_{\max}$ represent the upper and lower bound value of q_i, respectively.

The quality parameters of service cluster are assigned with an interval value. For example, if a service cluster is set with a quality parameter $q = \{\mathrm{RT}, <, [0.001, 0.1], \mathrm{ms}\}$, it means that the minimum response time of its component service is lower than 0.001 ms and the maximum response time of its component service is not more than 0.01 ms. The quality parameters of a service cluster can be classified as two categories: positive parameters and negative parameters. A higher value of positive parameter indicates a better service quality, for example, the response ratio and reliability. Conversely, the lower the value of negative parameter is, the better the service it can provide. For example, service price and failure rate are the negative parameters. The symbol Q^+ represents the set of positive quality parameters while Q^- is used to denote the set of negative quality parameters in this study.

Input: the responding service cluster S, the current binding Web service s and the quality of service request $Sr.Q$;
Output: the substituted Web service s_e for s;
Atom_evolve$(S, s, Sr.Q)$
(1) { if ($|S.S_{\_rw}| > 1$) Select a Web service randomly from $\{S.S_{\_rw} - s\}$ to assign as s_e;
(2) else { $S_E = \varnothing$;
(3) for each $ws \in S.S_w$
(4) if ($ws.Q \rhd Sr.Q$) $S_E = S_E \cup \{ws\}$;
(5) $O_E = Sr.O - S.Oc$;
(6) $i = 1$;
(7) for each $o \in O_E$
(8) $\{WS\_o_i = \{ws \mid ws \in S_E \wedge f(o_i) \in ws.I\}; i{+}{+}; \}$
(9) for $i = 1$ to $|O_E|$
(10) $ws_i = \{ws \mid \max(\text{rank}(ws)) \wedge ws \in WS\_o_i\}$;
(11) $s_e = ws_1 \oplus ws_2 \oplus \cdots \oplus ws_{|OE|}$;
(12) if ($s_e.Q \rhd Sr.Q$) return(s_e);
(13) else return(False); }

Algorithm 1: Isomorphic evolution for atomic Web service.

3. Isomorphic Evolution for Atomic Web Service

Isomorphic evolution for atomic Web service means to find a substituted service for the evolving Web service. The substituted one must be equivalent to the evolving one in both interface parameters and service quality. We divide the isomorphic evolution for atomic Web service into two patterns, the simple evolution and the merging evolution.

The responding service collection $S_{\_rw}$ contains more than one Web service and we can directly select another service to replace the evolving one in simple evolution. When there is no Web service that can substitute the evolving one in $S_{\_rw}$, the merging evolution needs to be performed. Two or more Web services in the responding service cluster are employed to combine as a new service and the new service is utilized to replace the evolving one in the merging evolution.

Compared with the merging evolution, simple evolution is with a less time cost. So it is priorly executed in the service request and response architecture based on service clusters. Merging evolution is carried out only when the simple evolution cannot satisfy the evolution request. The service request is formally defined as a 4-tuple $S_r = \{R_{\text{des}}, I, O, Q\}$ in this study. R_{des} is the function description of service request and the meaning of other items can refer to the definition of Web service (see Definition 1). To match interface parameter and evaluate the service quality are main operations in the service substitution and isomorphic evolution. Now we give some definitions about interface matching and service quality evaluation. Meanwhile, the isomorphic evolution algorithm is also provided in Algorithm 1.

Definition 3 (semantic equivalence). Let S_1 and S_2 be two concept nodes of the same ontology tree; S_p is the nearest common ancestor node of S_1 and S_2. $p(S)$ is used to represent the quantity ratio of child nodes of S and all the nodes of the ontology tree. The semantic compatibility is denoted as $SeSim(S_1, S_2)$.

$$SeSim(S_1, S_2) = \frac{2 \times \log p(S_p)}{\log p(S_1) + \log p(S_2)}. \tag{1}$$

Two concepts S_1 and S_2 are called the semantic equivalence, written as $S_1 \leftrightarrow S_2$, if $SeSim(S_1, S_2) \geq \delta$, where δ is a threshold value.

Let $P = \{p_1, p_2, \ldots, p_m\}$ be a group of parameters and p_i be the atomic parameter which cannot be decomposed. $|P|$ is used to denote the quantity of atomic parameters in P. The mapping functions $Type(p_i)$ and $Val(p_i)$ are utilized to represent the parameter type and value range of p_i, respectively. Meanwhile, we use $Type(p_1) \cong Type(p_2)$ to express that the data type of p_1 can match the data type of p_2. Based on the above elaboration, we classify the matching type of service parameters into three levels: subset parameter, isomorphic parameter, and value-covering parameter.

Definition 4 (parameter matching type). (1) P_1 is called the subset parameter of P_2, denoted as $P_1 \propto P_2$, if $|P_1| \leq |P_2|$ and $\forall p_i \in P_1, \exists p_j \in P_2$: $p_i \leftrightarrow p_j \wedge Type(p_i) \cong Type(p_j)$.

(2) P_1 is called the isomorphic parameter of P_2, denoted as $P_1 \Leftrightarrow P_2$, if $P_1 \propto P_2 \wedge P_2 \propto P_1$.

(3) P_1 is called the value-covering parameter of P_2, denoted as $P_1 \rhd P_2$, if $P_2 \propto P_1$ and $\forall p_i \in P_2, \exists p_j \in P_1$: $Val(p_i) \subseteq Val(p_j)$.

To provide a better user experience, we also need to evaluate service quality and select the optimal one to respond to the service request.

$$\begin{aligned} \text{rank}(s_i) &= \sum_{j=1}^{n} q_{ij} * w_j, \\ \sum_{j=1}^{n} w_j &= 1, \end{aligned} \tag{2}$$

$$q_{ij} = \begin{cases} \dfrac{q_j^{\max} - q_{ij}}{q_j^{\max} - q_j^{\min}}, & \text{if } q_j^{\max} - q_j^{\min} \neq 0; \\ 1, & \text{if } q_j^{\max} - q_j^{\min} = 0; \end{cases} \quad (3)$$

$$q_{ij} = \begin{cases} \dfrac{q_{ij} - q_j^{\min}}{q_j^{\max} - q_j^{\min}}, & \text{if } q_j^{\max} - q_j^{\min} \neq 0; \\ 1, & \text{if } q_j^{\max} - q_j^{\min} = 0. \end{cases} \quad (4)$$

Definition 5 (service quality score). Let $S = \{s_{w1}, s_{w2}, \ldots, s_{wm}\}$ be a group of Web services. Assume that each service in S has n quality parameters, and $[q_{i1}, q_{i2}, \ldots, q_{in}]$ is the quality vector of the service s_{wi}. The service quality score of s_{wi} is defined as follows.

There are two steps in computing service quality score: scaling phase and weighting phase. For a given service parameter $q \in Q^-$, formula (3) is adopted to scale the service quality vector. Formula (4) is utilized to scale the service quality vector for the service parameter $q \in Q^+$. In the weighting phase, formula (2) is used to compute the quality score of a Web service, where $w_j \in [0, 1]$ and w_j is the weight of q_j set by the users.

The first line in Algorithm 1 is used to handle the simple evolution for atomic Web service. In this case, the number of the responding service collections is greater than 1; that is, $|S.S_{\_rw}| > 1$, and we can randomly select a noncurrent Web service from $S.S_w$ to replace the current one. In the merging evolution, an empty service collection S_E is firstly initialized to store all the Web services that meet quality restraint of service request in $S.S_{\_rw}$ (lines (2) to (4)). The difference set of the output parameters in service request and the common output parameters in the responding service cluster is formally denoted as O_E. Acquire O_E and then for each parameter in O_E figure out the service collection which can provide the parameter, that is, building the collection WS_o_i in line (8). Finally, choose the Web service with the optimal service quality for each parameter in O_E and compose all selected web services as the service s_e in parallel. s_e can be returned as the evolved service if $s_e.Q$ is the value-covering parameter of the service request $Sr.Q$; otherwise the isomorphic service evolution is failed.

4. Isomorphic Evolution for Service Process

Since a service process is composed of multiple service clusters and the responding service collections of these service clusters may contain more than one Web service, many instances of the service process can be evolved by selecting different services in the responding service collections. These instances vary subtly from each other in the functions and service quality. Thus, the service process constructed based on service clusters is with a wide self-adapted ability in isomorphic evolution.

4.1. Formal Definition and Service Quality of Cluster Service Process. Cluster service process is used to briefly express the service process constructed based on service clusters in the following sections. Different from the isomorphic evolution for atomic Web service, the isomorphic evolution for service process needs to replace the evolved services from interface and quality in view of the whole process. To calculate service quality, cluster service net is introduced to formally describe the cluster service process and then a method to compute the service quality of service process is proposed based on the cluster service net.

Definition 6 (cluster service net). A cluster service net is defined as a 5-tuple $CSN = (\text{LPN}, i, o, \varphi, \Gamma, Q_F)$, where

(1) LPN is the process model of CSN while LPN is a logic Petri net with a single initial place and terminal place [29];

(2) $P = P_C \cup P_I \cup P_D$ is a set of places, where P_C is the set of control places to represent the logic transformation, P_I is the set of interior places, and P_D is the set of exterior places to make data interact with other service processes;

(3) $T = T_I \cup T_s \cup T_O$ is a set of transitions, where T_s is the set of service cluster transitions to represent the business operations of the service process. T_I and T_O are the set of logic input and output transitions, respectively, and they are used to control the business flow in the service process;

(4) φ: $T \to \Gamma$; φ is the mapping function between the set of transitions T and the set of their names Γ;

(5) Q_F is the set of service qualities of cluster service process.

Let Σ be a logic Petri net and x is a node of Σ; that is, $x \in P \cup T$; then ${}^\bullet x = \{y \mid (y, x) \in F\}$ is called the preset of x and $x^\bullet = \{y \mid (x, y) \in F\}$ is called the postset of x. The symbol π is introduced to get the preset and postset of the nodes in logic Petri net. The rules are defined as follows: $\pi_1(x) = {}^\bullet x$ and $\pi_2(x) = x^\bullet$. Now we give the definitions of the normal form of logic expression.

Definition 7 (normal form of logic expressions). $F = \{f_1, f_2, \ldots, f_n\}$ is a group of cluster service nets and t_o and t_i are the logic transitions to link F; that is, $\forall f_j$: $\pi_1(f_j.i) = \{t_o\}$ and $\forall f_j$: $\pi_2(f_j.o) = \{t_i\}$, where $1 \leq j \leq n$. The following types of logic expressions are called the normal form of logic expressions.

(1) $O_\vee$: $O(t_o) = f_1.i \wedge \neg f_2.i \neg \wedge \cdots \wedge \neg f_n.i \vee \neg f_1.i \wedge f_2.i \neg \wedge \cdots \wedge \neg f_n.i \cdots \vee \neg f_1.i \wedge \neg f_2.i \neg \wedge \cdots \wedge f_n.i.$

(2) $O_\wedge$: $O(t_o) = f_1.i \wedge f_2.i \wedge \cdots \wedge f_n.i.$

(3) $I_\vee$: $I(t_i) = f_1.o \vee f_2.o \vee \cdots \vee f_n.o.$

(4) $I_\wedge$: $I(t_i) = f_1.o \wedge f_2.o \wedge \cdots \wedge f_n.o.$

From the transformation rules between WS-BPEL and Petri net, we can know that any structure of a service process can be composed of sequence, parallel, alternative, and loop structures. The process structure constructed by only one of the sequence, parallel, alternative, and loop structures, that is, without nesting any other structure, is called onefold process block. Given n service clusters, the onefold process block of sequence, parallel, alternative, and loop structure is illustrated in Figure 2.

The alternative and parallel structures are identical except for their labeling logic expressions. Logic expressions of parallel structure are labeled with $O_\wedge$ and $I_\wedge$, respectively.

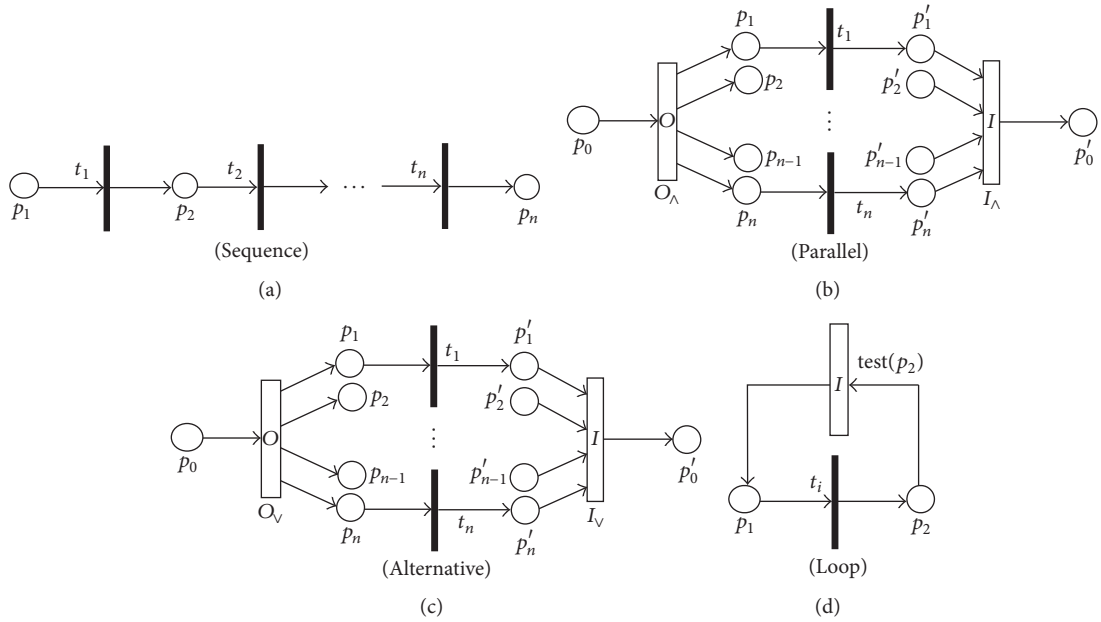

Figure 2: Onefold process block.

Table 1: Computing method of service quality for different structures.

Item	Sequence	Alternative	Parallel	Loop
$T(q)$	$\sum_{i=1}^{n} T(t_i)$	$\max\{T(t_i) \mid i = 1, \ldots, n\}$	$\max\{T(t_i) \mid i = 1, \ldots, n\}$	$K * T(t_i)$
$C(q)$	$\sum_{i=1}^{n} C(t_i)$	$\max\{C(t_i) \mid i = 1, \ldots, n\}$	$\sum_{i=1}^{n} C(t_i)$	$K * C(t_i)$
$R(q)$	$\prod_{i=1}^{n} R(t_i)$	$\min\{R(t_i) \mid i = 1, \ldots, n\}$	$\prod_{i=1}^{n} R(t_i)$	$K * R(t_i)$

Similarly, the first logic expression of alternative structure is marked with $O_\vee$ while the second one is marked with $I_\vee$.

Since the responding service collection may include several constituent services, service quality of the whole service process is determined by the extremum value of service quality in the responding service collection. For the positive parameters, we select the minimum service quality value of the Web services in the responding service collection to calculate the quality of service process. Meanwhile, the maximum service quality value of the Web services included in the responding service collection is adopted to compute the quality of service process for the negative parameters.

Let t be a service cluster including n Web services, q is a service quality parameter, and then $t.q = q.V_{\min}$ for $q \in Q^+$ and $t.q = q.V_{\max}$ for $q \in Q^-$. We use $T(q)$, $C(q)$, and $R(q)$ to represent the responding time, cost, and reliability of a Web service. Table 1 shows the method to compute the service quality for the cluster service process with sequence, parallel, alternative, and loop structure. By recursively using the proposed formulas in Table 1, we can calculate the service quality for the service process with any structure.

4.2. Isomorphic Evolution for Cluster Service Process. Isomorphic evolution for cluster service process is classified into two categories: dot isomorphic evolution and chain isomorphic evolution. The dot isomorphic evolution only seeks the substituted service in its responding service cluster for the evolving service and the other working services in the service process are not changed. In the chain isomorphic evolution, the evolving service and some of its associated web services will be replaced at the same time. The chain isomorphic evolution is restricted by the pattern of one-step successor for the high complexity in this study; that is, only the evolving service and one of its immediate successors can be replaced, and the other services in the service process are not changed.

4.2.1. Isomorphic Service Evolution for Dot Pattern. Comparison of interface parameters and reappraisal of service quality are two operations of isomorphic evolution for service process. Given a cluster service process, formally described as the cluster service net *CSN*, let *CSN.Qr* be the quality restriction from service request. The following operations are used in the isomorphic evolution for cluster service process.

(1) *InSCluster*(*ws*): acquire the service cluster which includes the Web service *ws*

(2) *InFlowStructure*(*ws*): acquire the onefold process block which includes the service *ws*

Input: the cluster service net CSN and the evolving service node n;
Output: the isomorphic evolution service cluster net $CSNe$ for CSN;
(1) $FlowEvo\_SingleDot(CSN, n, CSNe)$
(2) { Flag = Flase;
(3) $Q' = CSN.Qr - Q\_Structure(InFlowStructure(n) - n)$;
(4) $S = InSCluster(n)$;
(5) $n_e = Atom\_evolve(S, n, Q')$;
(6) $fe(n) = n_e$;
(7) if (n_e! = Null) $Flag$ = True;
(8) If ($Flag$ == True)
(9) $CSNe = Replace(CSN, n, n_e)$;
(10) Return $Flag$; }

Algorithm 2: Isomorphic evolution algorithm for dot pattern.

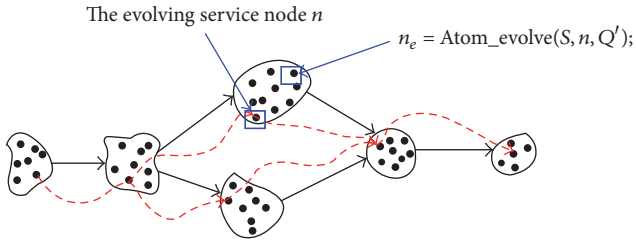

Figure 3: Dot isomorphic evolution for service process.

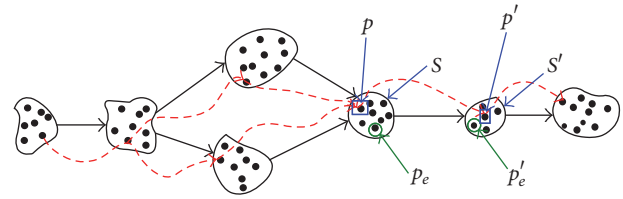

Figure 4: Chain isomorphic evolution for service process.

(3) $Q\_Structure(CSN)$: calculate the service quality of the cluster service net CSN

(4) $Replace(CSN, n, n')$: replace the service node n by n' in the cluster service net CSN

(5) $fe(n) = n_e$: fe is an evolution mapping function which assigns the service node n_e as the substituted service node of n.

In the isomorphic evolution for single node of dot mode (shown in Figure 3) (the procedure of dot isomorphic evolution can be found in Algorithm 2), we first obtain the onefold process block which includes the evolving service node n, and compute the difference value between the service quality restriction and the service quality of the onefold process block which has deleted the service node n (line (3)). Then, the difference value of service quality Q' is adopted as the new service quality restriction to perform the isomorphic evolution for atomic Web service of the service node n (lines (4) to (5)). If we get a valid substituted service n_e for the evolving service node n, the isomorphic evolution for single node of dot mode is successfully performed and the evolved service cluster net can be generated by replacing the service node n by n_e; otherwise the isomorphic service evolution is failed (lines (6) to (10)).

If there is more than one service node that needs evolving in the isomorphic service evolution with dot pattern, we need to execute the dot isomorphic evolution algorithm for several rounds. For each round of dot isomorphic evolution, we achieve the dot isomorphic evolution for one service node. The algorithm is repeatedly executed until all the service nodes have successfully performed the dot isomorphic evolution.

4.2.2. Isomorphic Service Evolution for Chain Pattern. We fist give an algorithm to handle the scenario of only one node performing the chain isomorphic evolution. For the isomorphic evolution method for single node of chain mode, one can refer to Algorithm 3. The proposed algorithm is used to achieve the isomorphic evolution of the service nodes which are not the starting and terminal ones in a service process.

Similar to the isomorphic evolution for single node of dot mode, we first calculate the new service quality restriction Q' of the cluster service net which had removed the evolving service node p. As shown in Figure 4, let p' be the successor node of p. The symbols S and S' are employed to represent the service clusters which include the service nodes p and p', respectively (lines (3) to (6)). We compute the potential input parameters by the mapping function f of S' and the output parameters of p'. If a new group of input parameters Iq which are different from the current input parameters of p' can be obtained, we build a virtual service node, written as q', where $q'.I = Iq$, $q'.O = p.O$, and $q'.Q = p.Q$. The isomorphic evolution for atomic Web service of the service node q' is performed in S' and the substituted node is denoted as p'_e. Sometimes we need to perform the isomorphic evolution repeatedly for the acquirement of node p'_e (lines (7) to (11)).

A virtual service node q will be customized once node p'_e was successfully obtained. The input parameters of p are set exactly the same as the output parameters of its precursor node. Meanwhile, the output parameters of p are set the same

Input: the cluster service net CSN, the evolving service node n;
Output: the isomorphic evolution service cluster net $CSNe$ for CSN;
(1) *FlowEvo_SingleChain*(CSN, n)
(2) { $p = n$;
(3) $Q' = CSN.Qr - Q\_Structure(InFlowStructure(p) - p)$;
(4) $S = InSCluster(p)$;
(5) $p' = nextnode(CSN, p)$;
(6) $S' = InSCluster(p')$;
(7) Select a parameter $Iq \in f(p'.O) - \{p'.I\}$;
(8) If $(Iq \neq \varnothing)$ $Iq = Iq \cup S'.Ic$;
(9) $q'.I = Iq$; $q'.O = p.O$; $q'.Q = p.Q$
(10) $p'_e = Atom\_evolve(S', q', q'.Q)$;
(11) if (p'_e == NULL) Go (7)
(12) $p'' = pirior(n)$;
(13) $q.I = p''.O$; $q.O = p'_e.I$;
(14) $p_e = Atom\_evolve(S, q, Q')$;
(15) if ($fe(n)$! = Null)
(16) $\{fe(n) = p_e$; $fe(p') = p'_e$;
(17) $N = \{n, p'\}$; $Ne = \{fe(n), fe(p')\}$;
(18) $CSNe = Replace(CSN, N, Ne)$;
(19) *Output(CSNe)*; return(True);}
(20) else return(False); }

Algorithm 3: Isomorphic evolution algorithm for chain pattern.

as the input parameters of p'_e. Then the isomorphic evolution for atomic Web service of node q is executed with the service quality restriction Q' in the service cluster S. The substituted service node for q is represented by p_e (lines (12) to (14)). We will evolve the cluster service net by replacing the service node n and its successor node p' once the service node p_e is successfully acquired; otherwise, the isomorphic service evolution is failed (lines (15) to (20)).

Isomorphic service evolution for multiple nodes of chain pattern means that there is more than one service node occurring in the isomorphic service evolution of chain mode. In the isomorphic service evolution for multiple nodes of chain pattern, we only need to perform the isomorphic service evolution for single node of chain pattern for all the evolving nodes one by one.

5. Simulation Experiment

Since no service clusters built based on application domain ontology can be found over the Internet, we can only verify the effectiveness of our proposed method by performing the simulation experiment. The aim of the experiment is to investigate whether the service discovery and evolution in the service clusters based SOA are more efficient than in the traditional SOA.

To obtain the service clusters, we first customize a large number of ontology concepts. The ontology concepts are selected from an ontology tree with the width of ten and the depth of eight. The numbers of the branches and leaf nodes are randomly generated. The total number of all the leaf nodes is more than 50000. By using the method to calculate the semantic similarity presented in Definition 3, the semantic similarity of any two Web services or service clusters can be easily acquired. The hardware environment in the simulation experiment is set as follows: CPU is the Intel core i5-4570 with 3.2 GHz, the memory is 8 G, and the OS is Windows 8. There are twenty-five computers that constitute a local area network. The data about service clusters is located equally among these computers. Each round of the simulation experiment was performed for twenty tests and the average value of these results was adopted as the final simulation results. The total number of the Web services in the experiment is 10000.

5.1. Service Discovery and Evolution of Atomic Web Service. Six rounds of experiments were performed while we compared the efficiencies of service discovery and evolutions between the traditional SOA and our three-tiered service request and response architecture oriented service clusters. The number N of service clusters in each round is 50, 100, 200, 500, 1000, and 2000, respectively. For each round, the number M of Web services in the service clusters is also randomly generated by $1 \leq M \leq 10 * (10000/N)$. As illustrated in Figure 5, the service discovery times of each round in service clusters based SOA are all lower than in the traditional SOA. However, the discovery time increases sharply with the granularity of service cluster turning small. Thus, we can get the conclusion that service discovery time is significantly affected by the granularity of service clusters.

The comparison of evolution times for atomic Web service between the traditional SOA and service clusters based SOA was shown in Figure 6. The substituted service needs searching in the whole service space in traditional SOA while the substituted one is sought only in the responding service

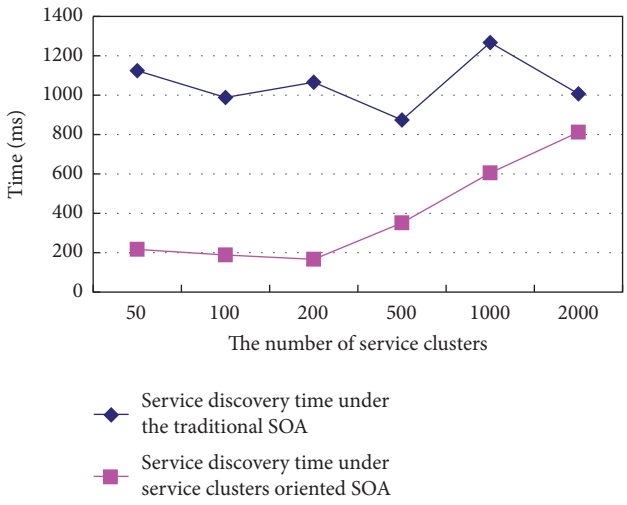

Figure 5: Comparison of service discovery times.

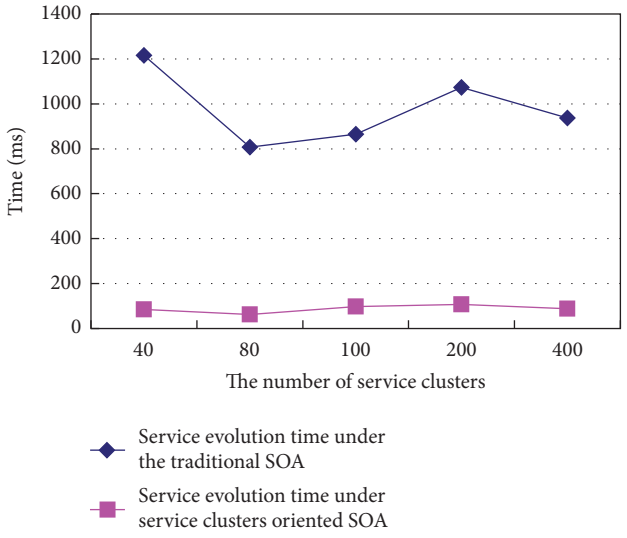

Figure 6: Comparison of service evolution times.

cluster. The searching space of the former is far greater than the latter. Thus, the evolution time is significantly reduced in the service clusters based SOA.

5.2. Isomorphic Evolution for Service Processes. Five rounds of experiments were performed while we compared the efficiencies of isomorphic evolution for service processes between the traditional SOA and service clusters based SOA. The number N of service clusters in each round is 40, 80, 100, 200, and 400, respectively. For every round, the number M of Web services in the service clusters is also randomly generated by $0.75 * 10000/N \leq M \leq 1.25 * 10000/N$.

Three cluster service processes with different flow patterns were constructed in the experiments. These service processes were composed of nine service nodes. We performed isomorphic evolution for service processes by one node and three nodes, respectively. For the results, one can refer to

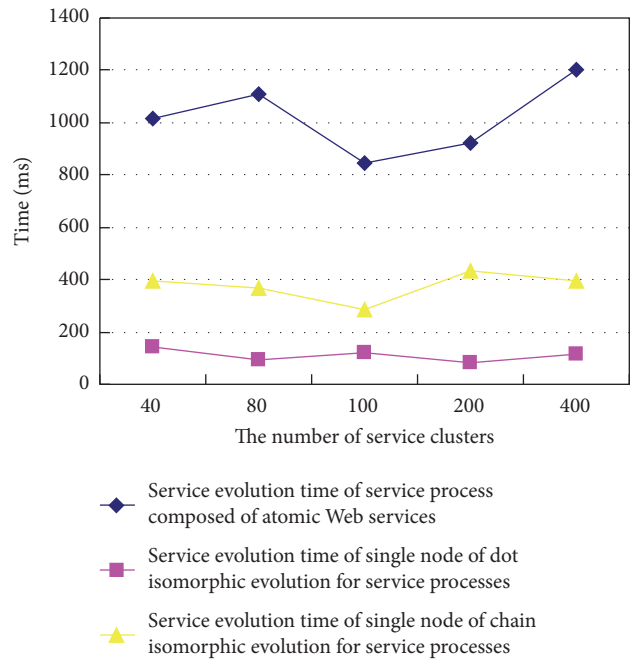

Figure 7: Comparison of evolution times for single node of isomorphic service evolution.

Figures 7 and 8. We can see that the efficiency of isomorphic evolution in the service clusters based SOA is higher than in the traditional SOA. Meanwhile, whether it is the dot pattern or the chain pattern, the isomorphic evolution for multiple nodes is less efficient than single node. For the isomorphic evolution of cluster service processes, the dot pattern is of a higher efficiency than the chain pattern. Thus, the dot isomorphic evolution for cluster service processes is taking precedence on execution.

6. Conclusions

In the traditional service request and response schema, fine granularity evolution of Web service is of a high cost because service substitution faces a large amount of similar services in functions and service quality. Particularly it will become more complicated when the isomorphic evolution for service process is executed for the adding of business logic. To reduce the difficulty in isomorphic service evolution, a three-tiered service request and response architecture oriented service clusters are proposed in this study. Service substitution is performed based on the service clusters and search space for alternative service is narrowed down to the average number of component services in the service clusters.

Meanwhile, isomorphic service evolution is carried out at the level of service clusters. Structures of original service processes need not be changed in most cases of isomorphic evolution since the response granularity is enlarged as a service cluster. Thus, the proposed method can easily achieve the isomorphic service evolution for the small search space in service substitution and very few structural changes in process evolution. We present the isomorphic evolution

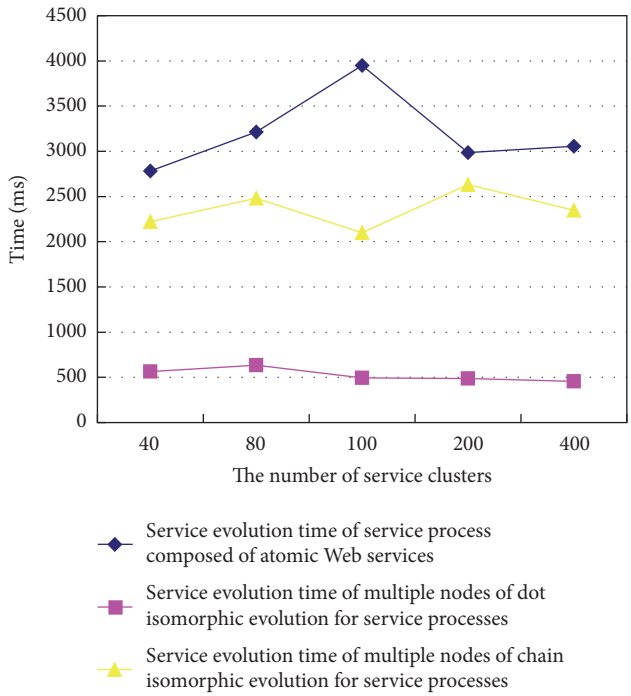

Figure 8: Comparison of evolution times for multiple nodes of isomorphic service evolution.

algorithms for the atomic Web service and service processes with different patterns. Simulation experiment was also provided to show the effectiveness and feasibility of the isomorphic service evolution mechanism based on service clusters. Relation between the granularity of service clusters and the time of isomorphic service evolution will be studied in the future work.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This study is supported by the Key Research Program of Shandong Province under Grant 2016GGX101031, the Promotive Research Fund for Young and Middle-Aged Scientists of Shandong Province under Grants BS2015DX010 and BS2015ZZ006, and the Natural Science Foundation of China under Grants 61273180 and 61472228.

References

[1] A. Botta, W. de Donato, V. Persico, and A. Pescapé, "Integration of cloud computing and internet of things: a survey," *Future Generation Computer Systems*, vol. 56, pp. 684–700, 2016.

[2] W. Tan, Y. Fan, A. Ghoneim, M. A. Hossain, and S. Dustdar, "From the Service-Oriented Architecture to the Web API Economy," *IEEE Internet Computing*, vol. 20, no. 4, pp. 64–68, 2016.

[3] A. L. Lemos, F. Daniel, and B. Benatallah, "Web service composition: A survey of techniques and tools," *ACM Computing Surveys*, vol. 48, no. 3, article no. 33, 2015.

[4] W. Zuo, A. N. Benharkat, and Y. Amghar, "Change-centric model for Web service evolution," in *Proceedings of the 2014 21st IEEE International Conference on Web Services, ICWS 2014*, pp. 712-713, Anchorage, AK, USA, July 2014.

[5] D. Rosaci and G. M. L. Sarné, "Recommending multimedia web services in a multi-device environment," *Information Systems*, vol. 38, no. 2, pp. 198–212, 2013.

[6] M. B. Blake and M. F. Nowlan, "A web service recommender system using enhanced syntactical matching," in *Proceedings of the 2007 IEEE International Conference on Web Services, ICWS 2007*, pp. 575–582, Salt Lake City, UT, USA, July 2007.

[7] L. Yao, Q. Z. Sheng, A. Segev, and J. Yu, "Recommending web services via combining collaborative filtering with content-based features," in *Proceedings of the IEEE 20th International Conference on Web Services (ICWS '13)*, pp. 42–49, IEEE, Santa Clara Marriott, Calif, USA, July 2013.

[8] X. Chen, Z. Zheng, X. Liu, Z. Huang, and H. Sun, "Personalized QoS-aware web service recommendation and visualization," *IEEE Transactions on Services Computing*, vol. 6, no. 1, pp. 35–47, 2013.

[9] G. Kang, M. Tang, J. Liu, X. F. Liu, and B. Cao, "Diversifying web service recommendation results via exploring service usage history," *IEEE Transactions on Services Computing*, vol. 9, no. 4, pp. 566–579, 2016.

[10] F. Messina, G. Pappalardo, D. Rosaci, C. Santoro, and G. M. L. Sarné, "A trust-aware, self-organizing system for large-scale federations of utility computing infrastructures," *Future Generation Computer Systems*, vol. 56, pp. 77–94, 2016.

[11] F. Messina, G. Pappalardo, C. Santoro, D. Rosaci, and G. M. L. Sarné, "A multi-agent protocol for service level agreement negotiation in cloud federations," *International Journal of Grid and Utility Computing*, vol. 7, no. 2, pp. 101–112, 2016.

[12] J. O. Gutierrez-Garcia and K.-M. Sim, "Self-organizing agents for service composition in cloud computing," in *Proceedings of the 2nd IEEE International Conference on Cloud Computing Technology and Science (CloudCom '10)*, pp. 59–66, Indianapolis, Ind, USA, November-December 2010.

[13] T. Nakano and T. Suda, "Self-organizing network services with evolutionary adaptation," *IEEE Transactions on Neural Networks*, vol. 16, no. 5, pp. 1269–1278, 2005.

[14] S. Wang, W. A. Higashino, M. Hayes, and M. A. M. Capretz, "Service evolution patterns," in *Proceedings of the 2014 21st IEEE International Conference on Web Services, ICWS 2014*, pp. 201–208, Anchorage, AK, USA, July 2014.

[15] D. Schumm, D. Dentsas, M. Hahn, D. Karastoyanova, F. Leymann, and M. Sonntag, "Web service composition reuse through shared process fragment libraries," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7387, pp. 498–501, 2012.

[16] A. F. Mohammad, J. Dargham, H. Mcheick, and A. T. Noor, "Software Evolution as SaaS: Evolution of Intelligent Design in Cloud," in *Proceedings of the 4th International Conference on Ambient Systems, Networks and Technologies, ANT 2013 and the 3rd International Conference on Sustainable Energy Information Technology, SEIT 2013*, pp. 486–493, Halifax, Nova Scotia, Canada, June 2013.

[17] V. Andrikopoulos, S. Benbernou, and M. P. Papazoglou, "On the evolution of services," *IEEE Transactions on Software Engineering*, vol. 38, no. 3, pp. 609–628, 2012.

[18] A. Goswami and R. P. Patel, "Service migration in cluster based cloud computing environment," in *Proceedings of the 2015 IEEE International Conference on Information Processing, ICIP 2015*, pp. 468–471, Pune, India, December 2015.

[19] M. Shiraz and A. Gani, "A lightweight active service migration framework for computational offloading in mobile cloud computing," *Journal of Supercomputing*, vol. 68, no. 2, pp. 978–995, 2014.

[20] G. Friedrich, M. Fugini, E. Mussi, B. Pernici, and G. Tagni, "Exception handling for repair in service-based processes," *IEEE Transactions on Software Engineering*, vol. 36, no. 2, pp. 198–215, 2010.

[21] Y. Liu, Y. H. Zhang, B. Zhang, M. W. Zhang, and Z. L. Zhu, "Analysis of service replaceability on behavior effect," *Journal of Computer Research and Development*, vol. 47, no. 8, pp. 1442–1449, 2015.

[22] W. Liu, Y. Du, M. Zhou C, and C. Yan, "Transformation of logical workflow nets," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 44, no. 10, pp. 1401–1412, 2014.

[23] Z. Wang and G. Cheng, "An approach to synergistic and dynamic service evolution in clouds," *International Journal of Cloud Computing*, vol. 4, no. 2, pp. 177–198, 2015.

[24] M. Fokaefs, R. Mikhaiel, N. Tsantalis, E. Stroulia, and A. Lau, "An empirical study on web service evolution," in *Proceedings of the 2011 IEEE 9th International Conference on Web Services, ICWS 2011*, pp. 49–56, Washington, DC, USA, July 2011.

[25] A. Jula, E. Sundararajan, and Z. Othman, "Cloud computing service composition: a systematic literature review," *Expert Systems with Applications*, vol. 41, no. 8, pp. 3809–3824, 2014.

[26] Q. Z. Sheng, Z. Maamar, L. Yao, C. Szabo, and S. Bourne, "Behavior modeling and automated verification of Web services," *Information Sciences*, vol. 258, pp. 416–433, 2014.

[27] C. Papagianni, A. Leivadeas, and S. Papavassiliou, "A cloud-oriented content delivery network paradigm: modeling and assessment," *IEEE Transactions on Dependable and Secure Computing*, vol. 10, no. 5, pp. 287–300, 2013.

[28] W. Liu, Y. Du, and C. Yan, "A web service discovery and composition method based on service classes," *Journal of Software Engineering*, vol. 7, no. 2, pp. 68–76, 2013.

[29] Q. Hu, Y. Du, and S. Yu, "Service net algebra based on logic Petri nets," *Information Sciences. An International Journal*, vol. 268, pp. 271–289, 2014.