Hindawi Publishing Corporation
The Scientific World Journal
Volume 2014, Article ID 793271, 11 pages
http://dx.doi.org/10.1155/2014/793271

Research Article

Dynamic and Quantitative Method of Analyzing Service Consistency Evolution Based on Extended Hierarchical Finite State Automata

Linjun Fan,[1] Jun Tang,[2] Yunxiang Ling,[1] and Benxian Li[3]

[1] *Science and Technology on Information Systems Engineering Laboratory, National University of Defense Technology, Changsha 410073, China*
[2] *Department of Telecommunications and Systems Engineering, Universitat Autònoma de Barcelona, 08202 Barcelona, Spain*
[3] *Department of Management Science and Engineering, Police Officer College of Chinese Armed Police Force, Chengdu 610213, China*

Correspondence should be addressed to Jun Tang; jun.tang@e-campus.uab.cat

Received 9 September 2013; Accepted 19 November 2013; Published 8 January 2014

Academic Editors: B. Johansson and P. Liu

This paper is concerned with the dynamic evolution analysis and quantitative measurement of primary factors that cause service inconsistency in service-oriented distributed simulation applications (SODSA). Traditional methods are mostly qualitative and empirical, and they do not consider the dynamic disturbances among factors in service's evolution behaviors such as producing, publishing, calling, and maintenance. Moreover, SODSA are rapidly evolving in terms of large-scale, reusable, compositional, pervasive, and flexible features, which presents difficulties in the usage of traditional analysis methods. To resolve these problems, a novel dynamic evolution model extended hierarchical service-finite state automata (EHS-FSA) is constructed based on finite state automata (FSA), which formally depict overall changing processes of service consistency states. And also the service consistency evolution algorithms (SCEAs) based on EHS-FSA are developed to quantitatively assess these impact factors. Experimental results show that the *bad reusability* (17.93% on average) is the biggest influential factor, the *noncomposition of atomic services* (13.12%) is the second biggest one, and the *service version's confusion* (1.2%) is the smallest one. Compared with previous qualitative analysis, SCEAs present good effectiveness and feasibility. This research can guide the engineers of service consistency technologies toward obtaining a higher level of consistency in SODSA.

1. Introduction

In recent years, service-oriented distributed simulations applications (SODSA) have become a major trend in modeling and simulation (M&S) field [1] mainly because of the enormous popularity of novel information technologies such as service-oriented architecture (SOA) [2], compositional simulation methods [3], cloud computing [4], internet of things [5], gridding, and Web service applications [1, 6, 7]. Owing to the usage of such technologies, SODSA is increasingly featured by service-oriented, composite, large-scale, ubiquitous, and flexible characteristics [8, 9]. For this reason, in order to ensure service consistency, traditional maintenance technologies which do not consider these novel attributes of SODSA exhibit some new difficulties (e.g., bad reusability, troubles of service composition, failures of service encapsulation and messages exchanges). It should be noted, however, that service consistency maintenance plays a critical role in the correctness and reliability of M&S. Therefore, it is crucial for software developers to make use of feasible technologies and methods to maintain service consistency. But prior to this step, we should first analyze the various influential factors that cause service inconsistencies, serving as guidance for the design of service consistency technologies with regard to a specific inconsistent factor. Traditional methods are mostly qualitative and experiential, and they do not consider the dynamic disturbances among factors in service's evolution behaviors such as producing, publishing, calling, and maintenance. Moreover, SODSA are rapidly evolving in terms of large-scale, reusable, composite, pervasive, and

flexible features, which presents difficulties in the usage of traditional analysis methods. To solve these problems, the main contributions of this paper are summarized as follows.

(i) We present a new service consistency evolution model, called extended hierarchical service-finite state automata (EHS-FSA), which is based on FSA theory and considers the dynamic disturbances among main inconsistency factors and simulates all transition behaviors and states of combined consistency attributes sets for each simulation service.

(ii) On the basis of EHS-FSA, we also develop two service consistency evolution algorithms (SCEAs) by running EHS-FSA and then calculating the number of factors occurrences affecting service inconsistency, which provides the quantitative analysis means of impact factors.

(iii) We carry out quantitative evaluation experiments to validate the feasibility and effectiveness of the service evolution model EHS-FSA and the statistical assessment algorithms SCEAs.

As a brief outline of the rest of the paper, Section 2 discusses the background and mechanism of service consistency evolution; Section 3 introduces the service consistency evolution model, including essential formal definitions and EHS-FSA; Section 4 proposes SCEAs and states their main principles; Section 5 gives the quantitative evaluation results and analysis of influencing factors by simulation methods; conclusion and the recommendations for future work are detailed in Section 6.

2. Related Work

Over the years, many maintenance techniques have been proposed in prior related studies to maintain all kinds of consistency states and ensure convenient utility of services in service evolution processes from the different emphasis points. Tian et al. [10] consider the software service reuse issues in ubiquitous environments and present a novel reuse approach in service consistency evolution processes, in which the reusable parts of existing services are extracted and directly utilized, and the missing functionalities-based index is then further implemented and built to accelerate the service reuse process. Sindhgatta and Sengupta [11] investigate the model transformation processes and the associated changing service design method in model-based development of SOA and introduce an extensible framework for tracing the dynamic evolution of model and model-based service. Frank and Karl [12] state consistency challenges of service discovery in mobile ad hoc networks and discuss the influencing factors on system efficiency that refers to transmission protocol, correctness of the delivered messages, message overhead, and even geographic distance between service providers and subscribers. Greenfield et al. [13] study the consistency maintenance problems of Web services, ensuring that services calls always be finished in consistent states despite failures and other exceptional events. Reference [13] addresses the relationship between internal service states, messages, and application protocols which facilitate the transformation from the problem of ensuring consistent outcomes into a protocol problem that can be easily validated by established service verification tools. There is also a great deal of other research achievements of service consistency evolution such as the studies in literatures [5, 14–18] that analyse the challenges and troubles and give some significative solutions. However, these solutions are usually adopted in such simulation environments where the number of simulation nodes is fixed and service developments are centralized and inflexible. With the emergence of SODSA, some new traits of software service developments are emerging, leading to traditional analysis methods being less applicable.

As for the above-mentioned former literatures [5, 14–18] that investigate service consistency in traditional distributed simulations compared with SODSA, there exhibit the following deficiencies in analyzing these influencing factors. (1) Only one or some of these factors are discussed, rather than most of enabled influential factors. (2) These studies only explain why these factors lead to service inconsistency, but do not specify the numerical extent which results in such inconsistent phenomena. (3) Dynamic perturbation mechanism between inconsistent factors in service running and their dynamically impact on all possible transitions of service consistency states are disregarded. (4) Existing researches focus mainly on inconsistency factors in special simulation phases or processes and do not synthetically investigate the issues that affect the overall software life cycle. Thanks to some major features of emerging SODSA such as large-scale attributes, dynamic assemblage, redundancy deployment, plug-in procedures, and flexible composition, the simulation nodes in SODSA are transformable, unpredictable, and therefore stochastic. These traits result in an increasingly unreliable and large number of services. Thus, the dynamic evolution mechanism of impact factors on service inconsistency should be exhaustively investigated, and inconsistent details in each simulation service should be carefully monitored in the whole simulation course. By using this analysis way, we can provide an actual and exact picture of the importance of such factors and can enable the excellent design of technologies for maintaining service consistency when it comes to the influencing factors, which facilitate the avoidance of wrong simulation results. When software engineers take into account all influencing factors in such complicated and dynamic simulation environments, they certainly encounter some difficulties in actual analysis, design, and running of service. In this paper, thus, we restrict the analysis to focus on some important inconsistency factors and temporarily ignore secondary ones.

Emerging automata theories [19], graph-based approaches [20], and formal methods [21] in software engineering supply many valuable solutions for describing and analyzing such complicated state changes and interactive behaviors in distributed systems, which provide us with constructive ideas. In particular, a finite state automaton (FSA) [22] has been widely used in the analysis and behavior modeling for various practical moderate-scale systems. However, in large-scale complex distributed systems, as it involves the complexity of sizes, behaviors, states, and

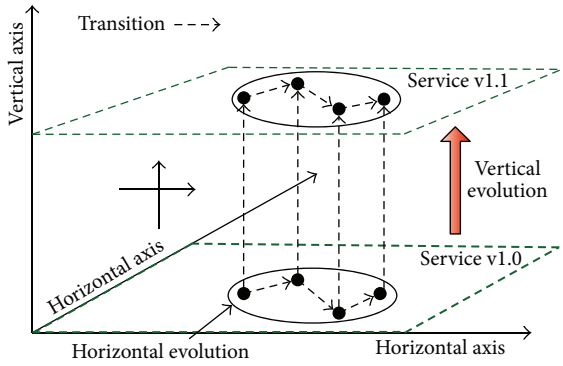

Figure 1: Three-dimensional view of service evolution.

Table 1: Inconsistency influencing events and their categorization.

Event	Type	Consistency meaning
SC	ES	Feasibility and integrity
ID	ES	Diversity of interface description
RS	NES	maintenance of several backups for the same service
WE	ES	Encapsulation correctness of web service based on models
VM	ES	Version control for the same service
MI	NES	Service communication without failure
SM	NES	Services' matching degree between publishers and subscribers
SR	NES	Whether the same service can be called by multiple other services

properties, a general FSA is not easy to be used to formally describe such systems. The extended hierarchical FSA (EHFSA) [23] that lets a state itself as an FSA and can depict complex state transitions is deepening and an expansion of FSA, avoiding the state space explosion and improving the efficiency of state transition. Considering the multifarious consistency evolution behaviors of atomic and composite services in large-scale SODSA, as well as the layered distribution of service grid, it is feasible that we use EHFSA to describe the consistency state evolution of service behaviors under compositional SODSA. In order to obtain quantitative analysis results, we also design two statistical evaluation algorithms to detect impact factors of service inconsistency.

3. Analysis of Service Consistency Evolution

3.1. Evolution Mechanism. Considering the actual situations in current SODSA, it is rare that there always exists an appropriate service exactly satisfying the user requirements. Because the popular tendencies of SODSA are that service evolution caters to continually changing requirements and services are dynamically combinated and rapidly updated. Hence, some macrolevel factor refers to service inconsistency, such as the diversity of service behaviors, the complex structures of the service itself, and the service's function overlap, can be observed in service-oriented simulations, which are the focus of service evolution in this research. Based on the above-mentioned factors, we mainly pay attention to the global consistency state evolution of services, monitoring the service state transition triggered by the impact factors, investigating the occurrence mechanism of inconsistent phenomena, and counting the influence rates of these factors.

In this paper, according to the evolution direction of services, the service consistency evolution can be divided into two categories: horizontal and vertical. Horizontal evolution refers to the internal consistency state changes for the same version of services and vertical evolution refers to service consistency problems between the same service's different versions. For each single service, we can describe its evolution processes by a three-dimensional view as shown in Figure 1. Additionally, the events of service evolution also can be summarized into two types: essential (ES) and nonessential (NES), on the basis of the product life cycle of service. The ES-events include service composition (SC), interface description (ID), Web encapsulation (WE), and version management (VM), referring to the processes of service's production and maintenance, whereas the NES-events such as redundant storage (RS), message interaction (MI), search and matching (SM), and share and reusage (SR) refer to the service's application processes. These events are the important elements of services' consistency state evolution. Table 1 shows the detailed events or factors of service consistency evolution.

3.2. Inconsistency Factors. In this section some formal method-based concepts for modeling are first introduced. A service consistency evolution model EHS-FSA is then proposed based on FSA. The representation of Section 3.1 indicates that the consistency state of service evolution can be analyzed from macroscopical factors based on structures, behaviors, and functions of service, which consist of service interface protocols (*structure*), storage deployment (*structure*), service production and maintenance (*behavior*), message communication (*behavior*), share and reusage (*function*), matching between service publishers and subscribers (*function*), and other ones. For example, the failures of message interactions among different services can prevent service running due to the network delay, package loss, hardware or software troubles, different service may have a diversity of description styles such as HTTP, SOAP, WSDL, and XML [24], which makes services' calls and compositions more difficult.

In fact, service composition is very significant in SODSA. It means that one single service can be composed of different atomic services, in which the atomic service is function-simple and relatively independent, but this combinative service has a larger granularity and more applications [25]. The composition among different services can provide some value-added functions and meet the subscribers' requirements [25]. However, owing to the heterogeneous, distributed, and dynamic network environment, the service composition is affected by changes in communication mode, network block, denial attacks of service, infrastructure

failures, and other issues [24], making the compositional behaviors of service not easy.

Note that the occurrence of any inconsistency factors in software engineering must have certain statistical regularity and these factors causing service inconsistency are no exception. Exactly speaking, the factors' importance degree should have a certain ordinal and numerical relationship, for instance, which factor is the least influential factor?, which factor is the most influential?, and what are the impact proportions of these factors?, respectively. Our main target in this paper is solving these problems by using formal method and FSA theory.

4. Consistency Evolution Model

In this section some formal method-based concepts for modeling are first introduced. A service consistency evolution model EHS-FSA is then proposed based on FSA.

4.1. Definitions and Notations. We first define related concepts which are the background knowledge of proposed EHS-FSA method and its algorithm for factors quantification analysis.

Definition 1 (atomic service). We define $\mathrm{AtS} = \{\alpha_i \mid i \in N^*\}$ as the set of atomic services. α_i (abbreviated as α) is the smallest service unit in SODSA, which encapsulates several models into one kind of web service and can only complete a simple task.

Definition 2 (composite service). Several atomic services can be appropriately assembled into a larger granular service. Let $\mathrm{CoS} = \{\beta_i \mid i \in N^*, \beta_i = \langle \alpha_1, \alpha_2, \ldots, \alpha_k \rangle\}$ denote the set of compositional services where β_i (abbreviated as β) is a k-tuple meaning the composite array and k denotes the number of α.

We assume that a single α that can be used to establish many β is the basic service unit and some service activities such as publishing, accessing, and calling can be executed only after the happen of α's composition behavior.

Definition 3 (global service space). We define a logically transparent space $\mathrm{GSS} = \mathrm{AtS} \cup \mathrm{CoS}$ which consists of all α and β in SODSA system. For GSS, α and β in different physical nodes logically belong to the same node. That is, the service is transparent and seems to be deployed locally, although actually saved at geographically dispersed regions.

According to Definition 3, GSS is like a container in which each service interaction can be done locally and there donot exist the remote accesses and calls, and all service-related consistency state evolutions can be executed in GSS.

Assume that β can no longer be integrated into a higher level of service with other β after several α are assembled into it. That is, there are only two types of service in SODSA: α and β, indicating the number of service layer is 2.

Figure 2 shows a simple example of α's behaviors of composition and reusage and β's message exchanges in GSS. In Figure 2, symbol "①" denotes the combination of different

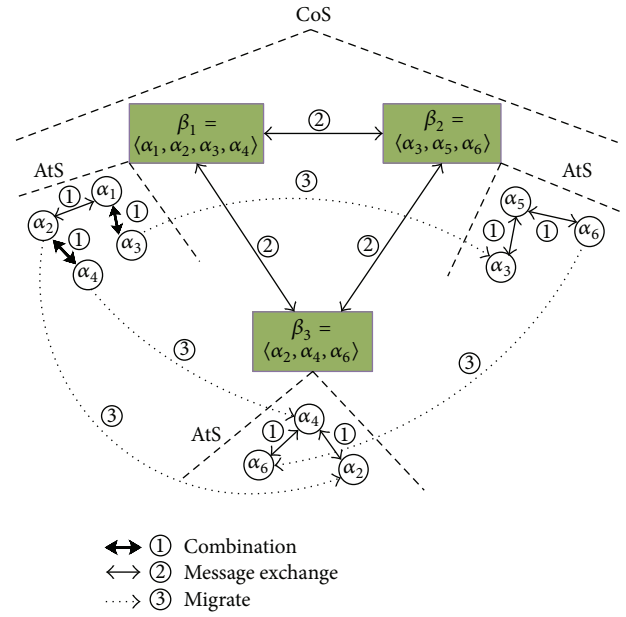

Figure 2: Example of service composition, reusage, and messages exchanges.

α, symbol "②" denotes message interactions between two business-related β, and symbol "③" denotes the migration of α, meaning the atomic services' share and reusage.

To simplify the problems and easily achieve formal descriptions, we consider only dominating influential events (or factors) on consistency state of services. These events include SC, WE, ID, SR, MI, and VM (as shown in Table 1).

Definition 4 (consistency attributes of AtS). Let a three-tuple $\mathrm{ATO}_\alpha = \langle \mathrm{VM}_\alpha, \mathrm{WE}_\alpha, \mathrm{SR}_\alpha \rangle$ denote the consistency-related properties set of α in GSS. The elements in ATO_α correspond to the before-mentioned events VM, WE, and SR, respectively, which should be carefully considered in maintaining the consistency state of α.

Definition 5 (consistency attributes of CoS). In GSS, use a four-tuple $\mathrm{COM}_\beta = \langle \mathrm{ID}_\beta, \mathrm{MI}_\beta, \mathrm{SC}_\beta, \mathrm{Aset} \rangle$ to denote β's consistency attributes set, where $\mathrm{Aset} = \mathrm{ATO}_{\alpha_1} \cup \mathrm{ATO}_{\alpha_2} \cup \cdots \cup \mathrm{ATO}_{\alpha_k}$, k is the number of current α in corresponding β, ID_β denotes the factor ID's consistency state, MI_β represents the factor MI's correctness, SC_β denotes the composition capability of the factor SC.

Seen as Definition 4, we can observe that all attributes of α are included in relevant β attributes but are just a subset of COM_β. Therefore, the consistency attributes of β can be divided into two levels: atomic layer and compositional layer. There are multiple α and β in GSS.

Definition 6 (consistency state array of AtS). For all t, use the state array $\mathrm{AtS}_\alpha^t = (\mathrm{vm}_\alpha, \mathrm{we}_\alpha, \mathrm{sr}_\alpha)$ (abbreviated as AtS_α^t) to denote the monitoring values of atomic service α's consistency status where t is the simulation time, the elements vm_α, we_α, and sr_α whose range is {true, false}, respectively, mapped

the events VM, WE, and SR in ATO_α. The logic term "true" denotes consistency, whereas "false" denotes inconsistency.

Definition 7 (consistency state array of CoS). Let the composited state array $\mathrm{CoS}^t_\beta = \langle(\mathrm{id}_\beta, \mathrm{mi}_\beta, \mathrm{sc}_\beta), \mathrm{AtS}^t_\alpha\rangle$ (abbreviated as CoS^t_β) denote β's monitor value of consistency states at any t where $(\mathrm{id}_\beta, \mathrm{mi}_\beta, \mathrm{sc}_\beta)$ is called the root array, and AtS^t_α is called the leaf array. The monitor values in CoS^t_β whose range is uniform to that in Definition 6 mapped the attributes in COM_β, respectively.

According to Definitions 6 and 7, AtS^t_α and CoS^t_β can quantitatively describe the consistency states of services. In the running of SODSA, we can use them for each α and β to monitor service state changes. Note that each α and β in GSS have eight such state arrays ($2^3 = 8$).

Give an example of consistency state arrays: at t, assume that the state arrays of α are $\mathrm{AtS}^t_{\alpha_1} = (0,1,1)$, $\mathrm{AtS}^t_{\alpha_2} = (1,1,1)$, $\mathrm{AtS}^t_{\alpha_3} = (1,1,0)$ and $\mathrm{AtS}^t_{\alpha_4} = (1,0,0)$. If α_1 is integrated with α_2 to produce β_1 and β_2 is composed of α_3 and α_4, then we have the state arrays of β $\mathrm{CoS}^t_{\beta_1} = \langle(1,0,1), (0,1,1)\cup(1,1,1)\rangle$ and $\mathrm{CoS}^t_{\beta_2} = \langle(1,1,0), (1,1,0)\cup(1,0,0)\rangle$.

Definition 8 (input events set of service states). Considering several primary consistency factors of α and β, we define the input events set of service states $I = AI \cup CI$, $AI = \{a_i \mid i = 1,2,\ldots,6\}$, and $CI = \{c_i \mid i = 1,2,\ldots,6\}$ where AI and CI are, respectively, the input set of state transition of α and β. The element a_i in AI refers to the following factors: version management confusion (a_1), encapsulation troubles (a_2), hard reusability (a_3), version maintenance (a_4), encapsulation repair (a_5), and reusability improvement (a_6). Similarly, c_i refers to these factors: different interface description (c_1), message communication failure (c_2), α's noncomposition (c_3), standardization of interface definition (c_4), successful message transmission (c_5), and α's composability amendment (c_6).

Definition 9 (output events set of service states). The consistency state transition of services can be triggered by input events, resulting in the output events whose set is denoted by $O = AO \cup CO$, $AO = \{d_i \mid i = 1,2,\ldots,6\}$, $CO = \{h_i \mid i = 1,2,\ldots,6\}$. AO and CO are the output set of state transition of α and β, respectively. For each d_i, we define the following output events of α: version inconsistency (d_1), α's internal inconsistency (d_2), nonreusability (d_3), version consistency (d_4), encapsulation correctness (d_5), good reusability (d_6). For each h_i, involving the following β's output events: inconsistent interface (h_1), message exchanges inconsistency (h_2), composition inconsistency (h_3), interface consistency (h_4), message interaction consistency (h_5), and composition consistency (h_6).

4.2. Extended Hierarchical Service-Finite State Automata. In this section, we construct an extended and hierarchical FSA to portray the dynamic evolution mechanism of service consistency states in SODSA.

Table 2: Transition lists of α's partial consistency states.

Current state	I	δ	O	Next state
(1, 1, 1)	a_1	$\delta((1,1,1), a_1)$	d_1	(0, 1, 1)
(1, 1, 1)	a_3	$\delta((1,1,1), a_3)$	d_3	(1, 1, 0)
(1, 1, 1)	a_2	$\delta((1,1,1), a_2)$	d_2	(1, 0, 1)
(1, 1, 0)	a_1	$\delta((1,1,0), a_1)$	d_1	(0, 1, 0)
(1, 1, 0)	a_6	$\delta((1,1,0), a_6)$	d_6	(1, 1, 1)
(1, 1, 0)	a_2	$\delta((1,1,0), a_2)$	d_2	(1, 0, 0)
(1, 0, 1)	a_1	$\delta((1,0,1), a_1)$	d_1	(0, 0, 1)
(1, 0, 1)	a_3	$\delta((1,0,1), a_3)$	d_3	(1, 0, 0)
(1, 0, 1)	a_5	$\delta((1,0,1), a_5)$	d_5	(1, 1, 1)
(1, 0, 0)	a_1	$\delta((1,0,0), a_1)$	d_1	(0, 0, 0)
(1, 0, 0)	a_5	$\delta((1,0,0), a_5)$	d_5	(1, 1, 0)
(1, 0, 0)	a_6	$\delta((1,0,0), a_6)$	d_6	(1, 0, 1)

Definition 10 (extended hierarchical service-finite state automata, EHS-FSA). We define service consistency evolution as a set of extended FSA, which is formally denoted by $AC = \{AC_1, AC_2, \ldots, AC_i, \ldots, AC_m\}$. AC_i is an extended FSA represented by the nine-tuple $\mathrm{AC}_i = \langle L, L_0, Q, Q_0, I, O, TS, \delta, \lambda\rangle$, where:

(i) $L = \{\mathrm{AtS}^t_\alpha \mid i = 1,2,\ldots,k\}$ and $Q = \{\mathrm{CoS}^t_\beta \mid i = 1,2,\ldots,k\}$ denote the finite and nonempty set of consistency states of α and β in AC_i, respectively,

(ii) $L_0 \subset L$ and $Q_0 \subset Q$ are the sets of all state arrays of initial α and β, respectively,

(iii) I and O denote the finite and nonempty set of service state transition's input events and output ones, respectively,

(iv) $\mathrm{TS} = \{\mathrm{ts} \mid \mathrm{ts} = (\mathrm{CoS}^t_{\beta_j}, \mathrm{AtS}^t_\alpha) \xrightarrow{I/O} (\mathrm{CoS}^{t'}_{\beta_j}, \mathrm{AtS}^{t'}_\alpha)\}$ represents the rule set of state transition of service consistency, that is the changes of state arrays $\mathrm{CoS}^t_{\beta_j}$ and AtS^t_α triggered by I from the interval t to t',

(v) $\delta : (Q, L) \times I \to (Q, L)$ is the state transition function of service evolution,

(vi) $\lambda : (Q, L) \times I \to O$ is the output event function.

In EHS-FSA, we have such transition functions $\delta(\mathrm{AtS}^t_\alpha, AI) = \mathrm{AtS}^{t'}_\alpha$, $\delta(\mathrm{CoS}^t_\beta, \mathrm{CI}) = \mathrm{CoS}^{t'}_\beta$, and so on. To simplify the transition rule, we assume that there only exists one input event in I and one output event in O for each state transition. According to Definition 10, there are partial state transition details for each α as shown in Table 2.

Figure 3 gives an illustrated way to portray α's state transition processes, in which the rule a_i/d_i denotes the input event and output event for each α's state change, t_i is the current system time, and the symbol "$\xrightarrow{a_i/d_i}$" represents the service state's evolution behavior. For example, the state array $(0,0,0)$ evolves into $(0,0,1)$ triggered by the rule a_6/d_6. Similarly, there are the similar state transition courses of root arrays in β.

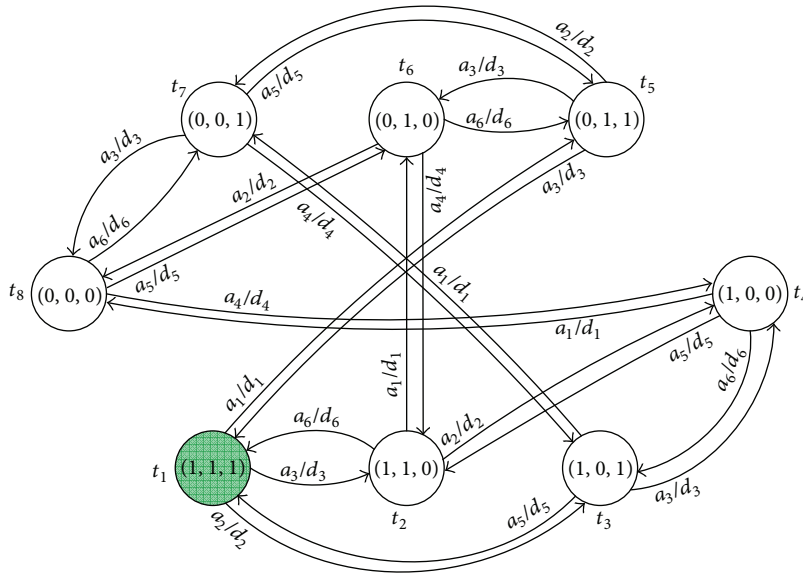

Figure 3: State transition view of α.

Definition 11 (states transition matrix of α). We define the following transition matrix:

$$\Pi = \left[\mathrm{AtS}_{\alpha}^{t}\right]_{m\times n}, \tag{1}$$

where $\mathrm{AtS}_{\alpha}^{t'} = \delta(\mathrm{AtS}_{\alpha}^{t}, a_i)$, $a_i \in AI$ $(i = 1, 2, \ldots, n)$, m denotes the number of all α states, and n denotes the amount of α's state input events that are listed by the sequence in AI. If the transition function $\delta(\mathrm{AtS}_{\alpha}^{t}, a_i)$ does not exist, $\mathrm{AtS}_{\alpha}^{t}$ in Π is represented by 0; contrarily, the state arrays $(1, 1, 1)$, $(1, 1, 0)$, $(1, 0, 1)$, $(1, 0, 0)$, $(0, 1, 1)$, $(0, 1, 0)$, $(0, 0, 1)$ and $(0, 0, 0)$ are, respectively, denoted by the values 1, 2, 3, 4, 5, 6, 7, and 8 in Π.

We can find that the number of α's actual states is 8 and the number of AI events is 6. Hence, the actual Π can be obtained as follows:

$$\prod = \begin{bmatrix} 5 & 3 & 2 & 0 & 0 & 0 \\ 6 & 4 & 0 & 0 & 0 & 1 \\ 7 & 0 & 4 & 0 & 1 & 0 \\ 8 & 0 & 0 & 0 & 2 & 3 \\ 1 & 7 & 6 & 0 & 0 & 0 \\ 0 & 8 & 0 & 2 & 0 & 5 \\ 0 & 0 & 8 & 3 & 5 & 0 \\ 0 & 0 & 0 & 4 & 6 & 7 \end{bmatrix}. \tag{2}$$

It can be observed that the contents of matrix Π are consistent with those of α in Table 2. Similarly, we can define the state transition matrix of β's root arrays as $Z = [\mathrm{CoS}_{\beta}^{t}]_{m\times n}$.

Definition 12 (occurrence probability of event set I). We use $P(a_i)$ and $P(c_i)$ to denote the occurrence probability of events a_i and c_i in I respectively. In our model, assume that $\sum P(a_i) = 1$, $P(a_1) = P(a_4)$, $P(a_2) = P(a_5)$ and $P(a_3) = P(a_6)$; similarly, $\sum P(c_i) = 1$, $P(c_1) = P(c_4)$, $P(c_2) = P(c_5)$, and $P(c_3) = P(c_6)$.

In general, the majority of α and β are difficult to be reused and combined, due to the heterogeneity of distributed systems and the diversity of modeling methods and tools and the knowledge differences in different application fields in SODSA. Therefore, the events a_3 and c_3 will become the very frequent activities. In addition to developing some new services, the events a_2 and c_1 also will occur usually when fusing the functions and applications in previous and old simulation systems into the new SODSA. From the statistical point of view, the operation running of software systems is basically stable, so the events a_1 and c_2 will occur occasionally.

On the basis of the above-mentioned actual situations, we assume that the occurrence probabilities of I will satisfy the following constraints in the dynamic evolution processes of EHS-FSA: $P(a_3) > P(a_2) > P(a_1)$, $P(c_3) > P(c_2) > P(c_1)$, $P(a_6) > P(a_5) > P(a_4)$, and $P(c_6) > P(c_5) > P(c_4)$.

5. Service Consistency Evolution Algorithms

In this section, two algorithms SCEA-α and SCEA-β to monitor service consistency transition activities are constructed, which focus on the occurrence of impact factors in SODSA on the foundation of executing EHS-FSA. Both SCEAs achieve a dynamic analysis of the influencing factors that lead to inconsistency of α and β. By statistically counting the number of effects of each factor in the operation running of EHS-FSA, ultimately, the importance of factors can be calculated to get a quantitative analysis results.

At the theoretical level, both SCEAs are dynamic running of EHS-FSA. That is, the dynamic transition processes of service consistency states are modeled by them in software simulation environment with time advancement, in which the continually changing behaviors and their consistency states referring to the elements of EHS-FSA can be monitored. At the application level, SCEAs can identify the factors' dynamic behaviors that cause α and β's inconsistency phenomena in SODSA and statistically draw the amount of each factor's occurrence to quantitatively analyze the evolution essence of service inconsistency.

```
(1)  Initialize N, K;
(2)  Num_VM, Num_SE, Num_SR←0; /* Counts of α's input events are initialized */
(3)  for i = 1 to N do
(4)    AtS^t_{α_i} ∈ L_0 ←(1, 1, 1); /* Initialize the state array of α */
(5)  endfor
(6)  for (k = 1; k++; k ≤ K) do
(7)    for each α_i ∈ AtS do
(8)      Get current state AtS^t_{α_i};
(9)      Find the rules ts ∈ TS at AtS^t_{α_i};
(10)     Select a_s ∈ AI according to its event probability;
(11)     Execute operation AtS^t_{α_i} ← δ(AtS^t_{α_i}, a_s); /* Trigger a state transition */
(12)     if a_s == a_1 then
(13)       Num_VM++; /* Count the number of event a_1 */
(14)     else a_s == a_2
(15)       Num_SE++; /* Add the number of event a_2 */
(16)     else a_s == a_3
(17)       Num_SR++; /* Add the number of event a_3 */
(18)     endif
(19)   endfor
(20) endfor
```

Algorithm 1: Pseudocode of SCEA-α algorithm.

The inputs of SCEA-α involve the following: (1) α's amount N and the cycle number K are initialized. (2) The counting values of inconsistent factors are initialized: $\text{Num}_{\text{VM}}, \text{Num}_{\text{SE}}, \text{Num}_{\text{SR}} \leftarrow 0$ (refer to the events a_1, a_2, and a_3, resp.). (3) Initialize α's state arrays: $\text{AtS}^t_{\alpha_i} \in L_0 \leftarrow (1, 1, 1)$. The outputs of SCEA-$\alpha$ include: the final numbers of factors occurrence that results in service inconsistency is determined: Num_{VM}, Num_{SE}, Num_{SR} and (i.e., the ultimate amounts of state transition events VM, SE, and SR). The detailed processes of SECA-α is described as the pseudocode in Algorithm 1.

The inputs of SCEA-β are as follows. (1) Initialize β and α's counts M, N, and the cycle number K. (2) The original values of inconsistent factors' counting are assigned: $\text{Num}_{\text{ID}}, \text{Num}_{\text{MI}}, \text{Num}_{\text{SC}} \leftarrow 0$ (refers to the events c_1, c_2, and c_3, resp.). (3) α's state arrays are initialized: $\text{AtS}^t_{\alpha_i} \in L_0 \leftarrow (1, 1, 1)$. (4) The initial composition schema of α: select several α randomly. (5) Initialize β's state arrays: $\text{CoS}^t_{\beta_j} \in Q_0 \leftarrow ((1, 1, 1), \text{AtS}^t_{\alpha_i} \cup \text{AtS}^t_{\alpha_l} \cdots)$. The outputs of SCEA-$\beta$ are: the ultimate amounts of β inconsistency states' factors: Num_{ID}, Num_{MI}, and Num_{SC}. Algorithm 2 gives the pseudocode description of SECA-β's detailed courses.

For the sake of the initial quantitative and dynamic exploration into service inconsistency factors and their evolution mechanism, therefore, ill-considered facets in SECAs design will inevitably appear, but the idea of using such novel analysis is an innovational and significative approach.

6. Quantitative Evaluation Experiment

6.1. Simulation Initialization. We implement the experimental evaluation using MATLAB R2009a on a PC with a Genuine Intel CPU T2400 (1.83 GHz) and 3.0 GB RAM, operated in Windows XP. We set the original number of α as 2–4 stochastically. Actually, owing to the small difference in the total number of α that minimally affects the algorithms' outputs when its count exceeds 50, the number of initial α could have been 5, 6, or 7 too. For example, if the total number of α is 156 or 157, the proportions of inconsistency factors (or events) remain at the same level when α's count is 155 in simulation experiments.

Table 3: The number of factors occurrences affecting service inconsistency.

K	a_1	a_2	a_3	c_1	c_2	c_3
100	1	5	18	3	10	13
200	2	11	37	5	17	28
300	4	14	50	11	31	40
400	6	18	74	15	38	49
Average (250)	3.25	12	44.75	8.5	24	32.5

At the initial evolution moment, several atomic services (α) are randomly composited into one β. The importance of each inconsistency factor (i.e., Num_{VM}, Num_{SE}, Num_{SR}, Num_{ID}, Num_{MI}, and Num_{SC}) can then be determined. To get the trustier results, we deploy four groups of experiments to execute the algorithms SECA-α and SECA-β where the total cycle numbers of them are 100, 200, 300, and 400. In addition, the selection of the input events in evolving EHS-FSA depends on the probability and constraints of the events occurrence defined in Section 4.2. For the exactly and credibly statistical analysis, all the results of the four experiments are averaged.

6.2. Results and Analysis. Table 3 shows that the experimental counting results on the effects of factors on service consistency which were derived by taking a different K and

```
(1)  Initialize M, N, K;
(2)  Num_ID, Num_MI, Num_SC←0; /* Counts of β's input events are initialized */
(3)  for i = 1 to N do
(4)    AtS^t_{α_i} ∈ L_0 ←(1, 1, 1); /* Initialize the state array of α */
(5)  endfor
(6)  Select 2–4 α_i stochastically which are assembled into β_j;
(7)  for j = 1 to M do
(8)    CoS^t_{β_j} ∈ Q_0 ← ((1, 1, 1), AtS^t_{α_i} ∪ AtS^t_{α_i} ···); // Initialize the β's state array
(9)  endfor
(10) for (k = 1; k++; k ≤ K) do
(11)   update AtS^{k+1}_{α_i} ← AtS^k_{α_i} by Algorithm 1;
(12)   for each β_j ∈ CoS do
(13)    Get current state CoS^k_{β_j};
(14)    Find the rules ts ∈ TS at CoS^k_{β_j};
(15)    Select c_s ∈ CI according to its event probability;
(16)    Execute operation CoS^{k+1}_{β_i} ← δ(CoS^k_{β_i}, c_s); /* Transition happens */
(17)    if c_s == c_1 then
(18)      Num_ID++; /* Add the number of event c_1 */
(19)    else c_s == c_2
(20)      Num_MI++; /* Count the number of event c_2 */
(21)    else c_s == c_3
(22)      Num_SC++; /* Add the number of event c_3 */
(23)    endif
(24)   endfor
(25) endfor
```

Algorithm 2: Pseudocode of SCEA-β algorithm.

running the programs of SECA-α and SECA-β. According to the collected data in Table 3, the impact ratios of service inconsistency events can be manually calculated, as shown in Table 4, whose data are a more accurate revelation.

To facilitate our experimental analysis in a more obvious manner, the statistical results in Table 3 are exhibited using the bar charts in Figures 4 and 5(b). Figures 4(a)–4(d) illustrate the continually rising times of each influence factor's occurrence while the algorithms' cycle number K, respectively equals 100, 200, 300, and 400, which imply the importance degree for service inconsistency, as shown in Table 4. It can be observed from Figure 4 and Table 4 that the order of these factors' importance degree is listed as follows.

(1) $K = 100$: a_3 (18%) > c_3 (13%) > c_2 (10%) > a_2 (5%) > c_1 (3%) > a_1 (1%).

(2) $K = 200$: a_3 (18.5%) > c_3 (14%) > c_2 (8.5%) > a_2 (5.5%) > c_1 (2.5%) > a_1 (1%).

(3) $K = 300$: a_3 (16.7%) > c_3 (13.3%) > c_2 (10.3%) > a_2 (4.7%) > c_1 (3.7%) > a_1 (1.3%).

(4) $K = 400$: a_3 (18.5%) > c_3 (12.25%) > c_2 (9.5%) > a_2 (4.5%) > c_1 (3.75%) > a_1 (1.5%).

From the listed-above sequences, we can observe that factor a_3 is the biggest influential one leading to inconsistent states of atomic service α and factor a_1 exerts minimal effect on α's consistency states. These sequences also indicate that the order of influencing proportion at different cycle times of SECA-α and SECA-β is the same despite the ratios' tiny differences.

Table 4: The impact rates of service inconsistency factors.

K	a_1	a_2	a_3	c_1	c_2	c_3
100	1%	5%	18%	3%	10%	13%
200	1%	5.5%	18.5%	2.5%	8.5%	14%
300	1.3%	4.7%	16.7%	3.7%	10.3%	13.3%
400	1.5%	4.5%	18.5	3.75%	9.5%	12.25%
Average (250)	1.2%	4.93%	17.93%	3.24%	9.58%	13.12%

In the following parts, we discussed what and why are the actual situations of the influencing rates of factors on service consistency evolution, as indicated by the experiment outcomes in Tables 3 and 4 and Figure 4. We combined some actual software development examples related to distributed service deployment in local area network (LAN) or wide area network (WAN) and evaluations results in this paper to discover some regular behaviors and give significative guidance for actual service consistency maintenance.

(1) The factors with very high impact (ranked 1 and 2) are α's bad reusability (a_3) and α's noncomposition (i.e., several atomic services are not composited into one larger service β) (c_3). This is mainly for the reasons: firstly, there is a great deal of incompatibilities in interfaces and deployment styles as fusing the old distributed simulation systems into the current SODSA; secondly, there exist many different development

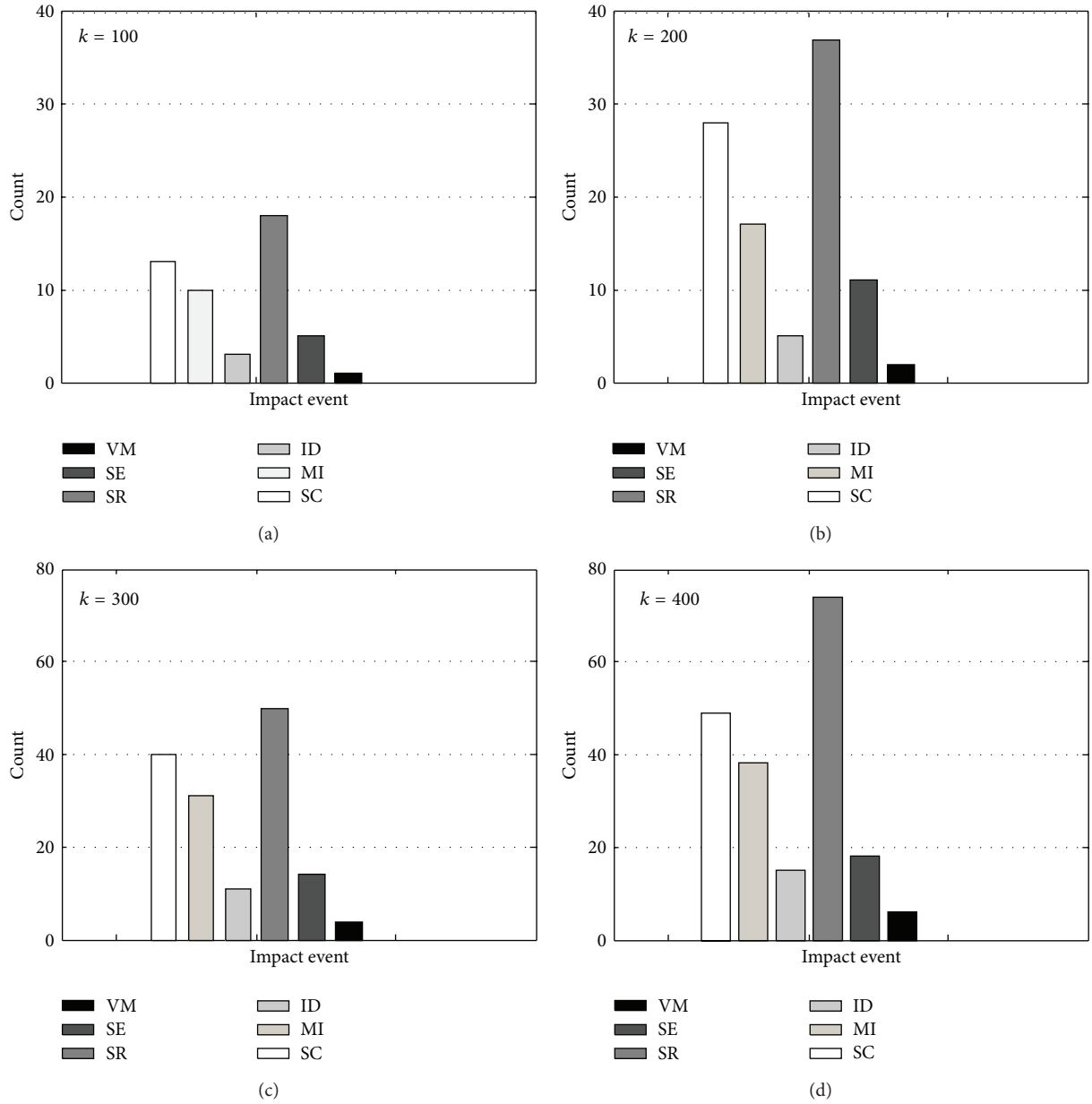

Figure 4: Comparison of the importance of different impact events.

tools such as C++, JAVA, MATLAB, Simulink, LABVIEW, and so forth; thirdly, the produced services are from different modeling domains such as electrics, communication system, machine design, hardware, and software. Such heterogeneity and diversity are prevalent, increasing the difficulty of service combination and reusability. These situations are based on objective facts in SODSA deployment and are not easy to be changed nowadays.

(2) The third important factor (ranked 3) is β's message communication failure (c_2). From the perspective of service application layer, we can say that c_2 is actually the most important influential factor on services inconsistency because in current or future SODSA, *message* is the most popular medium for service communication, and the message interactions between services are frequent and also can be easily blocked, due to the services' enormous amount, the continually entry and exit of simulation nodes, the instability of distributed network, and so on.

(3) The fourth and fifth factors with moderate impact (ranked 4 and 5) are α's encapsulation failure (a_2) and β's interface description difference (c_1), respectively. This is mainly because α's encapsulation and β's interface description are the basis of service calls, which occupy an important position in simulation service developments and directly affect such services behaviors like composition, reusage, interaction, and so on.

(4) The confusion of version management (a_1) is the smallest influential factor (ranked 6). In general, the management mechanism of simulation services is usually effective and stable once it has been established.

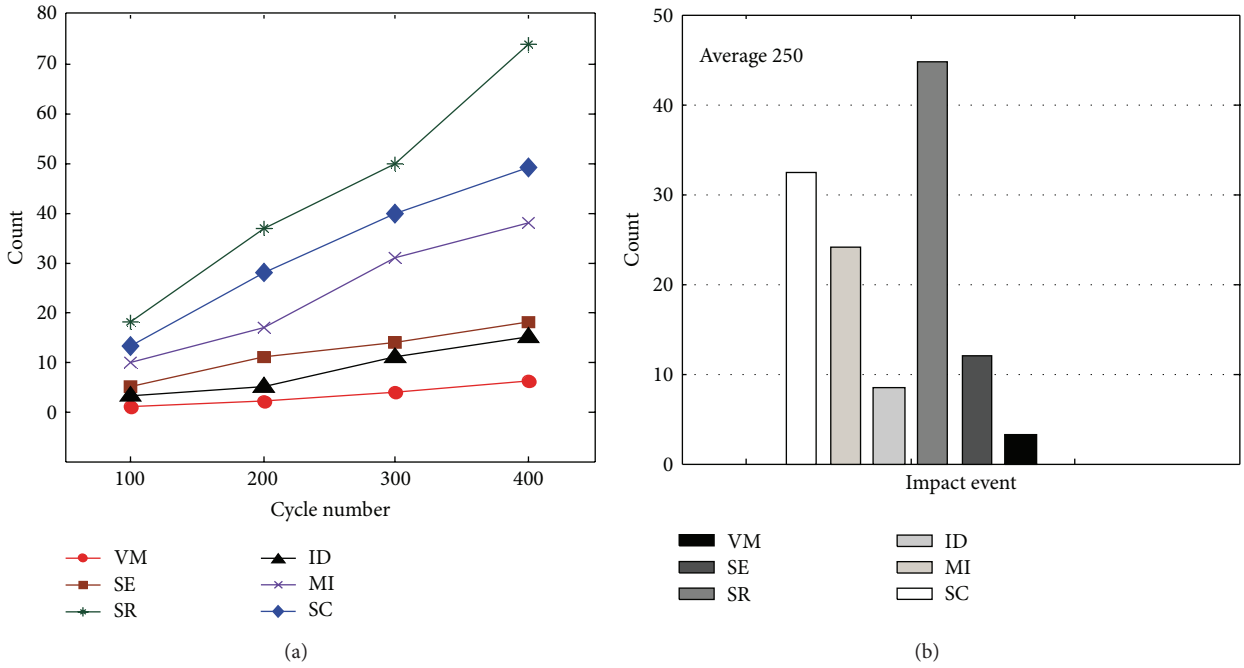

FIGURE 5: (a) Changing trends of factorial effects. (b) Average importance of factors at four cycle times.

Figure 5(a) indicates that the impact of each factor on service inconsistency states is continuously strengthened with the increasing cycle number K in SECA. We can also observe from Figure 5(a) that event a_3 has the greatest increasing effect on α, events c_3 and c_2 exert a greater increasing influence on β, whereas events a_2 and c_1 have a very small changing extent to affect α and β's consistency states, respectively, and a_1 has the smoothest changes whose effect on α is the smallest. On the basis of the average statistics at four cycle times, Figure 5(b) depicts the influencing factors' bar comparison by which the sorting of these factors and their impact rate is a_3 (17.93%) > c_3 (13.12%) > c_2 (9.58%) > a_2 (4.93%) > c_1 (3.24%) > a_1 (1.2%). Obviously, this order is consistent with the results in Figure 4 and each factor's impact rates at different cycle times only have a tiny difference, which improve the reliability of our experiment.

The above analysis results can provide an important concern on overcoming the service inconsistency risks in the design, development, publishing and subscription, operation running and maintenance of services in SODSA. To say the least, our experimental results may not completely reflect the actual influencing mechanism of service inconsistency factors, but most of them are in line with the actual situation. After all, in service evolution processes, the behaviors are complex, the attributes are various, and the system size is unpredictable, and so on. Hence, we must make some reasonable assumptions and self-defined rules by which the simulation evaluation can be finished effectively.

7. Conclusion

In this paper we propose an extended hierarchical finite state automata EHS-FSA and its two subalgorithms (SCEA-α and SCEA-β) to monitor the consistency states changes of α and β in SODSA. Based on theoretical and macroscopic perspective, EHS-FSA can formally portray the dynamic impact mechanism of service inconsistency behaviors and attributes in terms of reusability, composition, message exchange, service encapsulation, and so on. The presented SCEA aims to achieve a quantitative analysis of the inconsistency factors in service evolution for compositional SODSA, by running the EHS-FSA automata.

This study represents our preliminary attempt to introduce a novel analysis method of inconsistency factors which is completely different from previous ones. Our quantitative evaluation experiments show that EHS-FSA and SCEA are feasible, effective, advanced, and superior to traditional ones. The research achievements can offer theoretical and technical guidance for reducing service inconsistency states and improving the correctness of simulation services. In addition, our methods also can be applied to other domains such as the analysis and design of distributed-cooperative command posts that are deployed in future battlefields, complex distributed information systems based on network-centric warfare, embedded real-time systems.

Some future researches are as follows: (1) more inconsistency factors should be focused. (2) Some in-depth analysis of the disturbance mechanism among factors should be done. (3) The probability of event set I should be considered more carefully. (4) Related maintenance technologies of service consistency can be developed based on our experiment results.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this article.

Authors' Contribution

Linjun Fan and Jun Tang contributed equally to this work.

Acknowledgment

This research is partially supported by the National Natural Science Foundation of China (no. 61272336).

References

[1] W. Wang, W. Wang, Y. Zhu, and Q. Li, "Service-oriented simulation framework: an overview and unifying methodology," *Simulation*, vol. 87, no. 3, pp. 221–252, 2011.

[2] L. P. Chen, W. T. Ha, and G. J. Zhang, "Reliable execution based on CPN and skyline optimization for web service composition," *The Scientific World Journal*, vol. 2013, Article ID 729769, 10 pages, 2013.

[3] E. W. Weisel, *Models, Composability, and Validity*, Old Dominion University, Norfolk, Va, USA, 2004.

[4] Y. B. Yoon, J. Oh, and B. G. Lee, "The establishment of security strategies for introducing cloud computing," *KSII Transactions on Internet and Information Systems*, vol. 7, no. 4, pp. 860–877, 2013.

[5] M. A. Feki, F. Kawsar, M. Boussard, and L. Trappeniers, "The internet of things: the next technological revolution," *Computer*, vol. 46, no. 2, pp. 24–25, 2013.

[6] H. D. Kim, "Sharing e-learning object metadata using ebXML registries for semantic grid computing," *KSII Transactions on Internet and Information Systems*, vol. 2, no. 5, pp. 239–252, 2008.

[7] M. Malaimalavathani and R. Gowri, "A survey on semantic web service discovery," in *Proceedings of the International Conference on Information Communication and Embedded Systems (ICICES '13)*, pp. 222–225, Chennai, India, February 2013.

[8] S. Strassburger, T. Schulze, and R. Fujimoto, "Future trends in distributed simulation and distributed virtual environments: results of a peer study," in *Proceedings of the Winter Simulation Conference (WSC '08)*, pp. 777–785, Austin, Tex, USA, December 2008.

[9] L. Fan, Y. Ling, T. Wang, X. Zhu, and X. Tan, "Novel clock synchronization algorithm of parametric difference for parallel and distributed simulations," *Computer Networks*, vol. 57, no. 6, pp. 1474–1487, 2013.

[10] P.-W. Tian, Y.-X. Zhang, Y.-Z. Zhou et al., "A novel service evolution approach for active services in ubiquitous computing," *International Journal of Communication Systems*, vol. 22, no. 9, pp. 1123–1151, 2009.

[11] R. Sindhgatta and B. Sengupta, "An extensible framework for tracing model evolution in SOA solution design," in *Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications (OOPSLA '09)*, pp. 647–658, Orlando, Fla, USA, October 2009.

[12] C. Frank and H. Karl, "Consistency challenges of service discovery in mobile ad hoc networks," in *Proceedings of the 7th ACM International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM '04)*, pp. 105–114, Venice, Italy, October 2004.

[13] P. Greenfield, D. Kuo, S. Nepal, and A. Fekete, "Consistency for web services applications," in *Proceedings of the 31st International Conference on Very Large Data Bases (VLDB '05)*, pp. 1199–1203, Trondheim, Norway, September 2005.

[14] C. Dabrowski, K. Mills, and J. Elder, "Understanding consistency maintenance in service discovery architectures during communication failure," in *Proceedings of the 3rd International Workshop on Software and Performance (WOSP '02)*, pp. 168–178, Rome, Italy, July 2002.

[15] S. H. Ryu, F. Casati, H. Skogsrud, B. Benatallah, and R. S. Paul, "Supporting the dynamic evolution of web service protocols in service-oriented architectures," *ACM Transactions on the Web*, vol. 2, no. 2, article 13, pp. 1–39, 2008.

[16] M. P. Papazoglou, "The challenges of service evolution," in *Advanced Information Systems Engineering*, Lecture Notes in Computer Science, pp. 1–15, Springer, Berlin, Germany, 2008.

[17] J. Zou, X. Liu, H. Sun, and J. Zeng, "Live instance migration with data consistency in composite service evolution," in *Proceedings of the 6th World Congress on Services*, pp. 653–656, Miami, Fla, USA, July 2010.

[18] T. Weishäupl and E. Schikuta, "Dynamic service evolution for open languages in the grid and service oriented architecture," in *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, pp. 444–448, Pittsburgh, Pa, USA, November 2004.

[19] Z. Y. Dai, X. G. Mao, Y. Lei, Y. H. Qi, R. Wang, and B. Gu, "Compositional mining of multiple object API protocols through state abstraction," *The Scientific World Journal*, vol. 2013, Article ID 171647, 13 pages, 2013.

[20] H. Liu, Z. Zheng, W. Zhang, and K. Ren, "A global graph-based approach for transaction and QoS-aware service composition," *KSII Transactions on Internet and Information Systems*, vol. 5, no. 7, pp. 1252–1273, 2011.

[21] S. Bernardi, J. Merseguer, and D. C. Petriu, "Dependability modeling and assessment in UML-based software development," *The Scientific World Journal*, vol. 2012, Article ID 614635, 11 pages, 2012.

[22] L. Riano and T. M. McGinnity, "Automatically composing and parameterizing skills by evolving Finite State Automata," *Robotics and Autonomous Systems*, vol. 60, no. 4, pp. 639–650, 2012.

[23] R. Malik, M. Fabian, and K. Akesson, "Modelling large-scale discrete-event systems using modules, aliases, and extended finite-state automata," in *Proceedings of the 18th World Congress of the International Federation of Automatic Control (IFAC '11)*, pp. 7000–7005, Milano, Italy, August 2011.

[24] M. P. Papazoglou and W.-J. Van Den Heuvel, "Service oriented architectures: approaches, technologies and research issues," *The VLDB Journal*, vol. 16, no. 3, pp. 389–415, 2007.

[25] Q. Li, A. Liu, H. Liu, B. Lin, L. Huang, and N. Gu, "Web services provision: solutions, challenges and opportunities," in *Proceedings of the 3rd International Conference on Ubiquitous Information Management and Communication (ICUIMC '09)*, pp. 80–87, Suwon, Republic of Korea, January 2009.