Journal of Zhejiang University-SCIENCE C (Computers & Electronics)
ISSN 1869-1951 (Print); ISSN 1869-196X (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

Quantitative evaluation of model consistency evolution in compositional service-oriented simulation using a connected hyper-digraph[*]

Lin-jun FAN[1], Yun-xiang LING[1], Xing-tao ZHANG[1], Jun TANG[2]

(*[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 Autónoma de Barcelona, Barcelona 08202, Spain*)

E-mail: ljfan_nudt@163.com; yxling@tom.com; zhangxingtaolyx@163.com; 08202tangjun018@gmail.com

Received Apr. 8, 2013; Revision accepted Aug. 20, 2013; Crosschecked Dec. 16, 2013

Abstract: Appropriate maintenance technologies that facilitate model consistency in distributed simulation systems are relevant but generally unavailable. To resolve this problem, we analyze the main factors that cause model inconsistency. The analysis methods used for traditional distributed simulations are mostly empirical and qualitative, and disregard the dynamic characteristics of factor evolution in model operational running. Furthermore, distributed simulation applications (DSAs) are rapidly evolving in terms of large-scale, distributed, service-oriented, compositional, and dynamic features. Such developments present difficulty in the use of traditional analysis methods in DSAs, for the analysis of factorial effects on simulation models. To solve these problems, we construct a dynamic evolution mechanism of model consistency, called the connected model hyper-digraph (CMH). CMH is developed using formal methods that accurately specify the evolutional processes and activities of models (i.e., self-evolution, interoperability, compositionality, and authenticity). We also develop an algorithm of model consistency evolution (AMCE) based on CMH to quantitatively and dynamically evaluate influencing factors. Experimental results demonstrate that non-combination (33.7% on average) is the most influential factor, non-single-directed understanding (26.6%) is the second most influential, and non-double-directed understanding (5.0%) is the least influential. Unlike previous analysis methods, AMCE provides good feasibility and effectiveness. This research can serve as guidance for designers of consistency maintenance technologies toward achieving a high level of consistency in future DSAs.

Key words: Model consistency evolution, Factor quantification analysis, Connected hyper-digraph, Formal methods, Compositional service-oriented simulation
doi:10.1631/jzus.C1300089   **Document code:** A   **CLC number:** TP391.9

1 Introduction

We consider analysis and measurement issues related to the factors that influence the consistency of simulation models in distributed simulation applications (DSAs) with dynamically changing nodes and new computer technologies, such as grids, clouds, and service-oriented architectures. Distributed systems have rapidly evolved in terms of foundational architectures, usage scenarios, and operational mechanisms. This rapid development is due primarily to the popularity of new applications, such as grid and cloud computing (Wang *et al.*, 2011; Wu and Chen, 2012), service-oriented and pervasive computing (Cheun *et al.*, 2012; Yu *et al.*, 2013), and Internet of Things (Atzori *et al.*, 2010). Additionally, feasible middleware platforms based on DSAs have emerged; these platforms include grid platforms (Asiki *et al.*, 2009), service-oriented simulation middleware (Al-Jaroodi and Mohamed, 2012), and extensible modeling and

[*] Project supported by the National Natural Science Foundation of China (No. 61272336)

© Zhejiang University and Springer-Verlag Berlin Heidelberg 2014

simulation frameworks (Morse, 2005). These technologies reflect the future trends of DSAs. Consequently, the DSAs for these changing systems are increasingly characterized by large-scale, distributed, service-oriented, pervasive, and dynamic features (Strassburger *et al.*, 2008; Fan *et al.*, 2013). It should be noted, however, that a central component in model- or service-driven simulations is the model. Therefore, model consistency plays a critical role in the accuracy of simulation results (Dam and Ghose, 2011).

Many consistency maintenance techniques have been proposed to assure model consistency (Engels *et al.*, 2002; Yu and Vahdat, 2002; Ding *et al.*, 2005; Mens *et al.*, 2005; Yan, 2005; Guo *et al.*, 2012); these techniques are suitable for traditional simulation systems. They are also usually performed in simulation environments, where the number of nodes is fixed and balanced, and models are centrally developed. With the appearance of new DSAs, however, new characteristics of software modeling will emerge, causing traditional technologies to be less applicable to such DSAs. These features are summarized as follows (Dam and Ghose, 2011): (1) The projects of large-scale distributed simulation systems are inherently cooperative, in which numerous software practitioners are required, and excellent skills and collaborative communication are crucial to building authentic and mutually intelligible models. (2) Dispersed development and implementation in wide area networks for simulation projects result in software models being constructed at different geographical locations and implemented at various other geographical locations. (3) Given the size and complexity of systems, many complex models from different application domains exist, making interoperability and collaboration among models difficult.

Given this backdrop, maintaining consistency within simulation models during their evolution has become an impossible challenge for existing approaches. Therefore, new technologies or methods should be developed to maintain the consistency of simulation models in future DSAs. Prior to this step, the various factors that cause model inconsistency should first be analyzed to guide the design of consistency technologies with regard to a specific factor.

Previous literature (Foster, 2003; Zhou *et al.*, 2004; Tolk *et al.*, 2006; Beimel and Galanti, 2007; Lu *et al.*, 2007; Zhang *et al.*, 2009) that analyzes these factors suggests the following deficiencies: (1) Most methods for analyzing inconsistency factors (e.g., clock, syntax, semantics, and creditability) are static, experiential, and qualitative. (2) Related studies consider only certain factors, rather than multiple determinants or all possible factors. (3) The analysis methods are generally one-sided, taking into account only technical implementation and disregarding the management and conceptual consistency of model engineers. Existing studies that examine dynamic and quantitative consistency checking methods are limited (Law, 2005; Uhlig and Rude, 2009; Chen, 2011). These studies focus primarily on inconsistency factors in special simulation phases or processes, and do not synthetically investigate the problems that affect the overall software life cycle. The simulation nodes and models in DSAs are also unreliable and continually changing, making system size unpredictable. The large-scale attributes of systems result in an increasingly large number of models. Thus, the dynamic evolution mechanism of influence of factors on model inconsistency should be comprehensively analyzed, and inconsistent details in each model in the entire simulation processes should be carefully monitored. This approach will provide an actual picture of the importance of such factors, and enable the design of technologies for maintaining model consistency when it comes to the above-mentioned crucial factors. These features facilitate the avoidance of incorrect simulations. Engineers encounter difficulties in considering all influential factors in such complicated and dynamic simulation environments. In this paper, therefore, we restrict our analysis to primary influencing factors, such as self-evolution, interoperability, compositionality, and authenticity, and temporarily disregard secondary influencing factors.

Emerging formalisms and graph theory applications (Brandt *et al.*, 2009; Hermann *et al.*, 2011; Trollmann and Albayrak, 2011; Wei *et al.*, 2011; Weidlich *et al.*, 2011) in computer science provide numerous feasible solutions for describing and analyzing frequent interactions in many complex systems. A connected digraph with edge weights can be used to depict the complex and dynamic activities in a real-world system. For general consistency analysis, we define an extended directed hyper-graph, which is based on a directed and weighted graph, to formally describe the dynamic behaviors observed in model consistency evolution under compositional service-oriented simulations. To capture quantitative counting results, we introduce a statistical evaluation algorithm that detects model inconsistency factors.

The main contributions of this paper are as follows:

1. We propose an extended directed graph, called the connected model hyper-digraph (CMH), which formally describes evolutional activities (i.e., self-evolution, interoperability, compositionality, and authenticity) and their dynamic evolution mechanism.

2. Based on CMH, we design an algorithm of model consistency evolution (AMCE), which supports the quantitative and dynamic evaluation of impact factors.

3. We conduct quantitative measurement experiments to validate the feasibility and effectiveness of CMH and AMCE.

2 Background and analysis of model inconsistency

The prevalent trends of service-oriented compositional simulations are that modeling activities cater to continually changing requirements and models are dynamically and randomly assembled. Consequently, new characteristics such as complexity, dynamics, and uncertainty are observed in many modeling and simulation behaviors. As to consistency evolution in the model domain, various disturbing factors trigger inconsistency events. To simplify problems and easily construct formal descriptions, we consider only primary influencing factors and merge several factors into a single one when necessary. These factors include the uncertainty and complexity of model evolution in large-scale complex systems (Cicirelli *et al.*, 2011; Singh and Gracanin, 2012), the no-fill gap between established simulation models and actual physical models (Strassburger *et al.*, 2008), the difficulty in combination, reusability, and interoperability among simulation models (Zhou *et al.*, 2007), the diversity of semantic understanding under the same models for isomorphic or heterogeneous simulation units (Swinerd and McNaught, 2012), and different interface criteria (Ceranowicz *et al.*, 2012; Zhong and Huang, 2012). These factors can be summarized into two types: internal and external factors.

Internal factors can be abstracted as two attributes, namely the self-evolution and authenticity of models, whereas external factors can be converted into two attributes, namely interoperability and compositionality. Self-evolution pertains to the inherent consistency of the transition status of a model, and authenticity implies the degree of realism of a model as it is designed. Property compositionality refers to whether the interfaces of models can be seamlessly linked. The nature of interoperability indicates whether interactions and data communications among multiple models can be successfully conducted despite the inconsistent understanding of syntax and semantics. Fig. 1 shows the internal and external macro attributes of model consistency and the evolution mechanism of such inconsistency in the model domain.

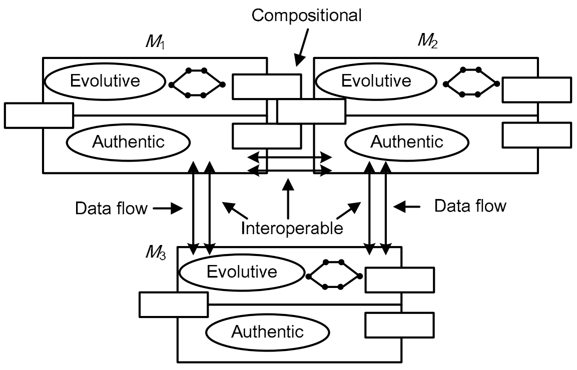

Fig. 1 Internal and external statuses of the model in consistency evolution

From a macro perspective, we consider the reliability and interactivities in model processes and implementation in modern DSAs. We investigate the overall activities that involve consistency issues in relation to authenticity, self-evolution, compositionality, and interoperability, as well as design evolution rules for modeling behaviors. Using formally descriptive methods and graph theory as the bases, we analyze the evolution mechanism of model consistency to develop an effective counting algorithm that can quantitatively measure inconsistent activities.

3 Model consistency evolution

This section provides the formal method-based notations and definitions for modeling. A directed model hyper-graph CMH is then presented to analyze the dynamic mechanism of model consistency evolution; graph theory is used in the analysis.

3.1 Notations and definitions

Before the construction of CMH, some related definitions and notations are established.

Definition 1 (Model fidelity) We assume that all the sub-models in DSAs can be composed into only a system-level model. Variable $\eta_{ij}\in[0, 1]$ is used to denote the degree of authenticity of simulation model j in node i to provide a contrast with objective physical models. In actual situations, node i indicates a computer in distributed networks. If a physical model and the corresponding simulation model are completely consistent, $\eta_{ij}=1$. If both models are fully inconsistent, $\eta_{ij}=0$. The fidelity of all the simulation models in DSAs is defined as

$$\rho=\sum_{i=1}^{n}\sum_{j=1}^{m_i} w_{ij}\eta_{ij}, \tag{1}$$

where $w_{ij}\in[0, 1]$ denotes the degree of importance of model j for node i, n denotes the number of nodes, and m_i denotes the total number of models in node i.

The skills of modeling practitioners and their ability to comprehend real models affect η_{ij}. However, such subjective effects are usually uncertain and stochastic. Thus, we assume that model j in node i is credible if $\eta_{ij}\geq 0.9$; otherwise, we classify this model as unreliable.

Definition 2 (Consistency entropy of self-evolution) Let $\mathrm{ES}_{ij}(t_0)$ denote the initial logical consistency status of model j in node i at t_0 before the initiation of model self-evolution. $\mathrm{ES}_{ij}(t)$ denotes the current logical consistency status at t as a model evolves. $\mathrm{ES}_{ij}(t)=1$ when model j at t is completely and logically correct, and $\mathrm{ES}_{ij}(t)=0$ when model j at t is logically incorrect. The time interval (in s) for the transition from the initial state to the current state is denoted by $\Delta t=|t-t_0|$. The consistency entropy of self-evolution is defined as follows:

$$E_{ij}(t)=\frac{|\mathrm{ES}_{ij}(t)-\mathrm{ES}_{ij}(t_0)|}{\Delta t}. \tag{2}$$

$\mathrm{ES}_{ij}(t)$ and $E_{ij}(t)$ can be abbreviated as ES and E, respectively. According to the definition above, we have $0\leq E\leq 1$. When $E=0$, the consistency states of a model in different self-evolution stages are similar, whereas when $E=1$, the current state of the model completely deviates from its initial state. In self-evolution, E reflects the change range of a model's intrinsic consistency state. That is, the extent of state change is small and model stability is strong at a small E. By contrast, the change in consistency state of a model has a large range and its stability is weak at a large value of E. Fig. 2 shows a transition example of consistency state during the self-evolution of a model.

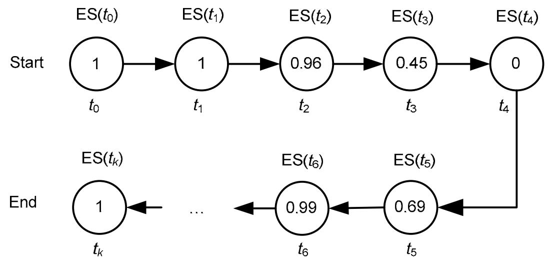

Fig. 2 Example of ES$_{ij}$(t) transition during the self-evolution of a model

In actual situations, modeling engineers are required to possess the necessary expertise through a series of training and learning programs before they are provided opportunities to work. Thus, most engineers are highly professional, highly skilled, and make minimal mistakes in the modeling processes. Most of the models they have created are logically consistent, with very few cases being inconsistent. A few inconsistent models have emerged because of engineer negligence, as well as poor physical and mental conditions during special projects. Given this situation, we assume that a state variable ES_{ij} obeys normal distribution.

Hypothesis 1 State variable ES follows a normal distribution $N(\mu, \sigma^2)$. Let ES fall in the interval $(\mu-\sigma, \mu+\sigma)$ to assume that the self-evolution of models is consistent. As an opposing condition, we consider a conflicting self-evolution. Thus, probability $P_{ij}\{|\mathrm{ES}-\mu|<\sigma\}$ is used to denote the consistency of self-evolution.

Based on Definition 2, let $P_{ij}=1-P_{ij}\{|\mathrm{ES}-\mu|<\sigma\}$ denote the state transition probability of the self-evolution of inconsistent models. The transition process is described as $\mathrm{ES}_{ij}\xrightarrow[\text{transfer}]{P_{ij}}\mathrm{ES}_{ij}'$. For instance, we can assign μ=1, σ=0.01 and calculate P_{ij}=0.3174 with reference to the standard normal distribution table.

Hypothesis 2 Simulation models cannot be both comprehensible and compositional in most cases because of differences in domain knowledge.

We define 'interoperability' as syntax and semantic interoperability among models, and 'understandability' as behavior that mutually occurs among models and not an inherent attribute of a single model. In this study we consider understandability as a special type of semantic interoperability.

Before we introduce Definition 3, it should be noted that one new node called v_i is being added to a set of already present nodes v_j ($j\neq i$).

Definition 3 (Understandability of the model graph) Let G=(V, A, L) be a connected digraph, where vertex $v_i\in V$ denotes simulation model $M_i\in M$ and arc set $A\subset V\times V$ such that (v_i, v_j)$\in A$ implies $i\neq j$, denotes the understandability among models. In addition, for any pair (v_i, v_j)$\in A$, the set L of notations $l_{i,j}$ or $l_{i,(j)}$ (or $l_{j,(i)}$, see remark below) for relationships between vertices discriminates between two (or three, see remark below) types of understandability: $l_{i,(j)}=v_i\rightarrow v_j$, meaning v_i can unidirectionally understand v_j; $l_{j,(i)}=v_j\rightarrow v_i$, meaning v_j can unidirectionally understand v_i; $l_{i,j}=v_i\leftrightarrow v_j$, meaning v_i and v_j can mutually understand each other. Then G=(V, A, L) denotes the understandability of the model graph.

Example 1 Fig. 3 shows three connection types for the vertices and edges of G. Figs. 3a and 3b depict single-directed understanding between M_i and M_j, and Fig. 3c provides an example of bidirectional comprehension.

A rare situation occurs when a newly added vertex v_i and existing vertices v_j in an evolving G are not understood bilaterally for some intermediate time while G is evolving, right after a new v_i has been added. We note this particular case as l_{ij}: $v_i\otimes v_j$ when ($v_i\rightarrow v_j\notin L$)$\wedge$($v_j\rightarrow v_i\notin L$).

The ability of engineers to achieve consistent semantic interoperability protocols or interfaces for DSAs is significant because it enables simulation systems to be efficiently and correctly implemented. Therefore, the definition of G depicts the behaviors and semantics of model interoperability in an abstract manner and lays the foundation for consistently interoperable evolution rules.

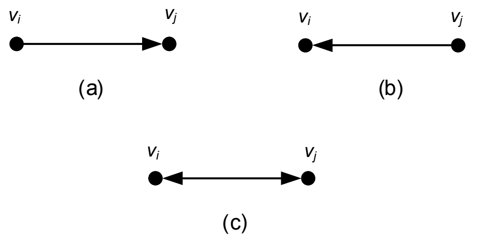

Fig. 3 Connection types for the vertices and edges of G
(a) Only M_i can understand M_j; (b) Only M_j can understand M_i; (c) M_i and M_j can understand each other

Definition 4 (Compositionality logic) Consider an understandable model graph G that does not take the directions of edges into account. If $\exists v_i$, $v_j\notin V$, $i\neq j$, $l\notin L$, we define the Boolean value b(v_i, v_j) or b(v_j, v_i) (abbreviated as b_{ij} or b_{ji}, $b_{ij}\notin${0, 1}) to represent the compositionality of M_i and M_j. Thus, the logistic assessments of all model combinations can be denoted by set $B=\{b_{ij}|v_i, v_j\notin V\}$. An intelligible relationship l between M_i and M_j is a necessary condition for the presence of b_{ij}. The value of b_{ij} denotes the compositionality of models, which means that M_i and M_j can be combined into one model if b_{ij}=1; they exist independently if b_{ij}=0. We therefore have $b_{ij}=1\Rightarrow l_{i,j}$: $v_i\leftrightarrow v_j\in L$.

However, converse derivation may not be established. Fig. 4 shows a simple example of compositionality logic in G.

v_i
1
v_j
v_i
0
v_j
(a)
(b)

Fig. 4 Example of compositionality logic in G
(a) M_i and M_j can be combined; (b) M_i and M_j cannot be combined

Example 2 Understandable model graphs G_1, G_2, G_3 are provided in three different simulation nodes, including their respective graphic illustration of compositionality logic. Fig. 5 shows that each model graph has five local simulation models.

Fig. 5 also illustrates that compositionality occurs only between vertices (or models) that belong to the same simulation node (i.e., G_i), whereas understandability may occur between vertices in different G_i's.

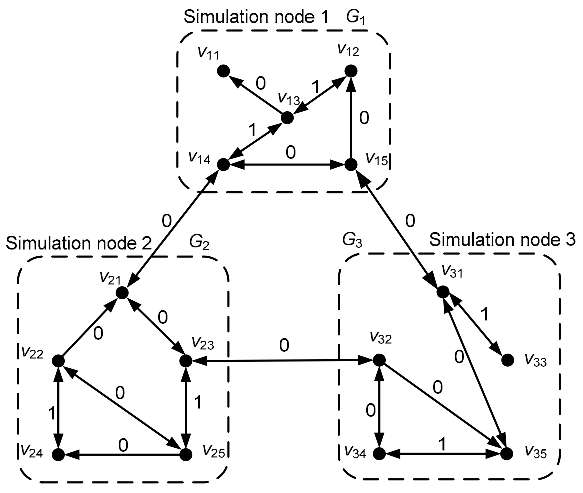

Fig. 5 Example of G_i and their compositionality logic

Definition 5 (Compositionality link) If $\exists v_i, v_j, v_q, v_k \in V$ in three-tuple $G=(V, A, L)$ and Boolean value set $B=\{b_{ij}|v_i, v_j \in V\}$ ($i \neq j \neq k$, $i \neq q \neq k$, j may be equal to q), let $K=(b_{ij}, \ldots, b_{qk})$ denote a Boolean logic link of compositionality. If $\forall b_{ij}=\ldots=b_{qk}=1$, K is a true logic link of compositionality, denoted as K^+, whereas $\exists b_{ij}$ or $b_{qk}=0$ if K is a false logic link of compositionality, denoted as K^-.

According to Definition 5, if K^+ but not K^- exists in G and, for example, the vertices in G are fully assembled, all models achieve combination consistency states. If K^- remains in G, then the models in G cannot be fully combined.

Definition 6 (Understandability link) Given $G=(V, A, L)$, if $\forall v_i, v_j \in V$, $i \neq j$, there always exists a two-way understandable link $\Gamma=(v_i, v_{i_1}, \ldots, v_{i_k}, \ldots, v_j)$, $i_k \neq i, j$, we may say that G is a fully understandable model graph and Γ is a completely accessible link, denoted as Γ^+: $v_i \leftrightarrow v_{i_1} \leftrightarrow \ldots \leftrightarrow v_{i_k} \leftrightarrow \ldots \leftrightarrow v_j$; if $\exists v_i', v_j', v_m', v_n' \in V$ ($i \neq j \neq m \neq n$), there is a reversible link $\Gamma^-=(v_i', v_{i_1}', \ldots, v_{i_k}', \ldots, v_j')$ and also there does not exist a bidirectional understanding relationship $l_{m,n}$: $v_m \leftrightarrow v_n$, then we call G a partially understandable graph and Γ^- a partially understandable link, denoted as Γ^-: $v_i' \leftrightarrow v_{i_1}' \leftrightarrow \ldots \leftrightarrow v_{i_k}' \leftrightarrow \ldots \leftrightarrow v_j'$.

Definition 6 indicates that G can realize an understandability-oriented consistency if and only if Γ^+ exists in G. Moreover, Definitions 5 and 6 show that when the vertices in G are interoperable and only K^+ exists in G, G can achieve compositionality-oriented consistency.

3.2 Connected model hyper-digraph

In this subsection, we propose a connected hyper-digraph for engineer modeling and model implementation. The relevant probabilities of event occurrence in model evolution are assumed.

A newly constructed simulation model that combines with G has three connections: (1) A newly jointed vertex can unilaterally access and understand the current connected vertex in G (Fig. 3a). (2) The existing connected vertex in G unidirectionally apprehends the newly added vertex (Fig. 3b). (3) Both vertices can be mutually understood (Fig. 3c). When the third connection occurs, the combined activities of the two vertices are characterized by being compositional or non-compositional. When the first or second connection happens, only non-compositionality occurs, indicating that the value of compositionality logic between vertices is 0.

The evolving hyper-digraph N based on G is defined to denote model consistency evolution.

Definition 7 (Evolution hyper-digraph) We define model consistency evolution as an eight-tuple $N=\langle G, G^0, U, H, L, B, f, t \rangle$, where $G=\{G_i|i=1, 2, \ldots, n\}$ denotes the evolution graphs set of all models, G^0 is the initial model graph, $U=\{\eta_{ij}|i=1, 2, \ldots, n; j=1, 2, \ldots, m\}$ denotes the variable set of authenticity degree close to that of real physical models, $H=\{E_{ij}(t)|i=1, 2, \ldots, n; j=1, 2, \ldots, m\}$ represents the logical consistency entropy set for the self-evolution of models, $L=\{l|l \in l_\tau, l_\upsilon, l_{\tau\upsilon}, v_\tau, v_\upsilon, \tau, \upsilon \in j\}$ denotes the set of understandability relationships in G, $B=\{b_{\tau\upsilon}|v_\tau, v_\upsilon \in V\}$ denotes the Boolean logic set of compositionality in G, f: $G_i \times (U, H, L, B) \rightarrow G_i^+$ is the evolution function of model consistency with $G_i \subset G_i^+ \subseteq G$, and t is the time axis of elements in N, which indicates that elements G, U, H, L, and B change as time passes.

Example 3 Fig. 6 shows a simple example of the state transition of N from t to t'; in t, let the element in $\{\eta_1, \eta_2, \eta_3, \eta_4\}$ denote the models' objective authenticity in G_i, the element in $\{E_1, E_2, E_3, E_4\}$ denote the consistency entropy in the models' self-evolution, and the element in $\{b_{21}, b_{31}, b_{23}, b_{34}, b_{14}\}$ represent the Boolean logic value of model compositionality. The figure shows the relation notations, i.e., $v_2 \rightarrow v_1$, $v_2 \leftrightarrow v_3$, $v_1 \leftrightarrow v_3$, $v_1 \leftrightarrow v_4$, and $v_4 \rightarrow v_3$. In t', for entry node v_5, understandability rule $\{v_1 \leftrightarrow v_5, v_4 \rightarrow v_5\}$ and combined rule $\{b_{15}, b_{45}\}$ are executed. The values of the authenticity measure and logical consistency evolution entropy are η_5 and E_5, respectively. Then, we obtain the state migration function using Definition 7:

$$f\colon G_i\times(\eta_5, E_5, v_1\leftrightarrow v_5, v_4\rightarrow v_5, \{b_{15}, b_{45}\})\rightarrow G_i^{+}.$$

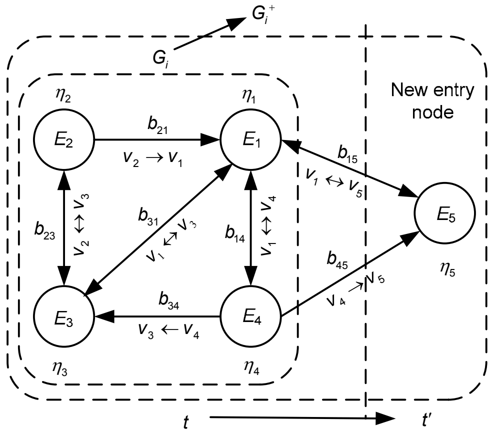

Fig. 6 Example of evolution hyper-graph *N*

Suppose that any two or three vertices randomly selected in current N can be connected with the newly joining node, whose probability of connecting existing vertices is stochastic. We then define the probability of the semantic intelligibility and combination of models.

Definition 8 (Probability distribution of understandability N) Let discrete random variable X be equal to x_k (k=1, 2, 3). Event X=x_1 indicates that a comprehensibility relation l_i: $v_i\rightarrow v_j$ exists when a new vertex is added. Similarly, event X=x_2 denotes the occurrence of the connection relation l_j: $v_j\rightarrow v_i$, and X=x_3 denotes that l_i and l_j simultaneously exist. The probability distribution of understandability N is defined as

$$P\{X=x_k\}=p_k,\ k=1, 2, 3. \tag{3}$$

Example 4 We obtain p_1=p_2, p_3=p_1+p_2 according to connection relation set L and modeling reality (models can be mutually understood in most cases; otherwise, the interaction among models is incomplete). We obtain p_1=p_2 because of the probability feature of variable X. Thus, we can obtain p_1=p_2=0.25 and p_3=0.5.

Definition 9 (Condition probability of compositionality N) Conditional probability (i.e., the occurrence probability of event $\{Y$=$y_i\}$ when event $\{X$=$x_k\}$ occurs) is used to denote the compositionality connection between models, defined as

$$P\{Y=y_i \mid X=x_k\}=\frac{P\{Y=y_i, X=x_k\}}{P\{X=x_k\}}, \tag{4}$$

where i=1, 2, k=1, 2, 3, $P\{X$=$x_k\}$ denotes the intelligible probability of models, and $P\{Y$=y_k, X=$x_k\}$ denotes the occurrence probability of behaviors that are both intelligible and compositional.

Example 5 Existing double-directed understandable connections in the evolution course of N are a necessary condition for combined behaviors. Therefore, the following are obtained according to Definitions 3 and 4 and the results are calculated using Definition 9:

$$P\{Y=0 \mid X=x_1\}=\frac{P\{Y=0, X=x_1\}}{P\{X=x_1\}}=1,$$
$$P\{Y=1 \mid X=x_1\}=0,\ \ P\{Y=0 \mid X=x_2\}=1,$$
$$P\{Y=1 \mid X=x_2\}=0,$$
$$P\{Y=0 \mid X=x_3\}=\frac{P\{Y=0, X=x_3\}}{P\{X=x_3\}}=0.5,$$
$$P\{Y=1 \mid X=x_3\}=0.5.$$

Definition 10 (Inconsistent event sets) Considering the attributes of model consistency such as authenticity, interoperability, compositionality, and self-evolution, we define the following inconsistent event sets: not matching reality (NMR), not single-directed understandability (NSU), not double-directed understandability (NDU), not combination (NC), and not consistent evolution (NCE), which are denoted by ME={NMR, NSU, NDU, NC, NCE}.

4 Algorithm of model consistency evolution

In this section, we design AMCE to model connection behaviors and focus on the occurrence of inconsistent events in DSAs based on the evolution of hyper-graph N and its rules (Section 3). AMCE conducts a dynamic analysis of the impact mechanism of factor disturbance that results in model inconsistency. It then statistically counts the effects of each primary factor in the evolution process of hyper-graph N. Ultimately, the importance of events can be obtained and the inconsistent factors in model evolution can be quantitatively analyzed.

The input conditions of AMCE are as follows:

1. Model graph G is initialized to G^0.

2. Initialize state attributes: calculate initial η_{ij}, $\mathrm{ES}_{ij}\leftarrow 1$, $b_{\tau\upsilon}\leftarrow 0$ ($\forall\tau$, $\upsilon\in j$).

3. The counts of influencing factors: initial $\mathrm{Num_{NMR}}$, $\mathrm{Num_{NCE}}$, $\mathrm{Num_{NSU}}$, $\mathrm{Num_{NDU}}$, and $\mathrm{Num_{NC}}$ are counted.

4. The number of cycles C is initialized.

The output results of AMCE are described as follows: The final counting values of influencing factors $\mathrm{Num_{NMR}}$, $\mathrm{Num_{NCE}}$, $\mathrm{Num_{NSU}}$, $\mathrm{Num_{NDU}}$, and $\mathrm{Num_{NC}}$ are determined.

The detailed processes of AMCE are described in Algorithm 1. At the logical level, Algorithm 1 is a dynamic implementation of evolving hyper-digraph N. That is, the dynamic evolution processes of model consistency are simulated by Algorithm 1 in simulation software developments. In software development with time advancement, the algorithm monitors the dynamic changes in the states of the elements in N (an eight-tuple) when a new model joins N. At the usage level, by simulating model consistency evolution, the algorithm also dynamically identifies the factors that affect model inconsistency and statistically draw the amount of each factor's occurrences to dynamically and quantitatively analyze the evolution mechanism of model inconsistency. The algorithm provided in this paper is only an initial exploration into model inconsistency factors and mechanism analysis in dynamic evolution processes; therefore, ill-considered aspects will inevitably appear. The idea of using dynamic and quantitative analysis, however, is a favorable exploratory approach.

Algorithm 1 AMCE algorithm

```
1  Initialize C, $G\leftarrow G^0$ // Initialize G and the cycle number
2  for each model j in $G_i$ do
3    Compute $\eta_{ij}$, $\mathrm{ES}_{ij}\leftarrow 1$, $b_{\tau\upsilon}\leftarrow 0$ $\forall\tau$, $\upsilon\in j$
     // Initialize the state attributes
     // Count inconsistent events
4    Count $\mathrm{Num_{NMR}}$, $\mathrm{Num_{NCE}}$, $\mathrm{Num_{NSU}}$, $\mathrm{Num_{NDU}}$, $\mathrm{Num_{NC}}$
5  endfor
6  for $\kappa\leftarrow 1$ to C do
7    for each model j in $G_i$ do
8      Execute the transfer $\mathrm{ES}_{ij}\xrightarrow[\text{transfer}]{P'_{ij}}\mathrm{ES}'_{ij}$
9      if $E_{ij}>\sigma$ then
10       $\mathrm{Num_{NCE}}+1$
         // Add the amount of self-evolution inconsistency
11     endif
12   endfor
13   Select any $v_\tau$, $v_\upsilon\in V$ ($\tau\neq\upsilon$) from G randomly
14   Execute $v_\kappa\xrightarrow[\text{connect}]{P\{X=x_k\}}v_\tau$, $v_\kappa\xrightarrow[\text{connect}]{P\{X=x_k\}}v_\upsilon$ by Eq. (3)
15   if $\exists v_\kappa\rightarrow v_\alpha$ or $v_\alpha\rightarrow v_\kappa$ ($\alpha\in\tau$, $\upsilon$) then
16     $\mathrm{Num_{NSU}}+1$, $b_{\kappa\alpha}\leftarrow 0$ // Count the number of NSU events
17     $\mathrm{Num_{NC}}+1$ // Add the number of NC events
18   endif
19   if $\exists v_\kappa\leftrightarrow v_\alpha$ then
20     Execute the conditional event by Eq. (4)
21     if $b_{\kappa\alpha}==0$ then
22       $\mathrm{Num_{NC}}+1$ // Count the number of NC events
23     endif
24   endif
25   if $\exists v_\kappa\otimes v_\alpha$ then
26     $\mathrm{Num_{NDU}}+1$ // Add the number of NDU events
27   endif
28   $G\leftarrow G+v_k$ // Update the model graph G
29   Compute $\eta_{i\kappa}$, $\mathrm{ES}_{i\kappa}\leftarrow 1$
       // Initialize the attributes of the joining node
30   if $\eta_{i\kappa}\geq 0.9$ then
31      $\mathrm{Num_{NMR}}+1$ // Count the number of NMR events
32   endif
33 endfor
```

5 Evaluation experiments

AMCE should be run by conducting simulation measurements instead of adopting the qualitative evaluation of experts and suggestions in the field. This approach will validate the performance (e.g., feasibility and effectiveness) of the proposed method and quantitatively evaluate inconsistency factors (or events).

5.1 Simulation setup

The proposed algorithm was coded by MATLAB R2009a and implemented on a PC with an Intel Dual-Core E5400 2.7 GHz CPU and 2.0 GB RAM, operated in Windows XP. Our evaluation was based on simulations, in which we set the original model graph G with four vertices (models) assigned to four different G_i's. The number of initial vertices could have been 2, 3, or 5, because the small difference in the total number of models minimally affects the output of the algorithm. For example, if the total number of models is 303, 304, or 305, the percentages of inconsistency events (factors) remain at the same level in simulations. In the initial evolution phase, the four vertices are randomly connected. The effect of each inconsistency factor (i.e., $\mathrm{Num_{NMR}}$, $\mathrm{Num_{NCE}}$, $\mathrm{Num_{NSU}}$, $\mathrm{Num_{NDU}}$, and $\mathrm{Num_{NC}}$) can then be determined. We also arranged four groups of experiments to run the algorithm; that is, the total cycle numbers of AMCE are 100, 200, 300, and 400, respectively. In addition, the connected vertex in evolving hyper-digraph *G* for a new node entering current *G* is selected according to the probability of the event occurrences defined in Section 3.2. For accurate statistical counting, all the results of the four experiments are averaged to enable reasonable and credible analysis and evaluation.

5.2 Summary of results

The experimental statistical results on the effects of factors on model consistency were derived by repeated running of AMCE with different maximum numbers of cycles. Table 1 shows the numbers as well as percentages of the five types of events that cause model inconsistency.

Table 1 The numbers and percentages of events leading to model inconsistency

κ	NC	NSU	NMR	NCE	NDU
100	35 (35%)	26 (26%)	21 (21%)	14 (14%)	4 (4%)
200	67 (33.5%)	53 (26.5%)	44 (22%)	27 (13.5%)	9 (4.5%)
300	103 (34.3%)	79 (26.4%)	58 (19.3%)	43 (14.3%)	17 (5.7%)
400	132 (33%)	108 (27%)	79 (19.75%)	61 (15.25%)	20 (5%)
Average (250)	84.25 (33.7%)	66.5 (26.6%)	50.5 (20.2%)	36.25 (14.5%)	12.5 (5%)

To exhibit our experimental results in a more intuitive manner, the data in Table 1 is also shown as bar charts and curve graphs in Figs. 7 and 8, respectively. Figs. 7a–7d illustrate the impact rates of model inconsistency events as the numbers of cycles equal 100, 200, 300, and 400, respectively. Fig. 7 and Table 1 indicate that the degree of importance of the factors follows the order NC>NSU>NMR>NCE>NDU in all the four cases. Event NC is the most influential factor for model inconsistency and event NDU exerts the minimal effect. Fig. 7 also shows that the sequence of importance degrees at different numbers of cycles is consistent and stable despite the proportion accounted for by different impact factors or their individual sequence.

A crucial requirement is the analysis of the actual effects of the dynamic disturbance mechanism of factors on model consistency evolution, as indicated

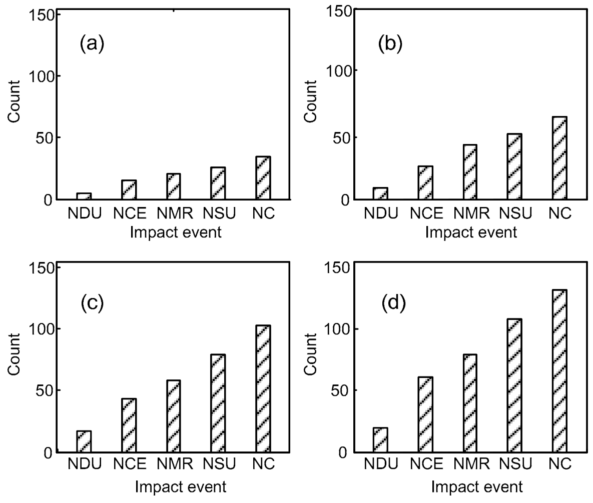

Fig. 7 Comparison of the importance degrees of different influencing factors
(a) κ=100; (b) κ=200; (c) κ=300; (d) κ=400

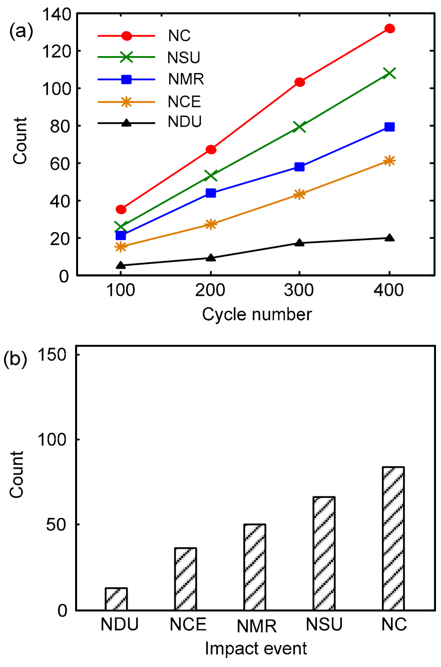

Fig. 8 Changing trends of factorial effects (a) and average importance degrees of the five types of events (b)

by the percentages given in Table 1. Some practical reasons and situations related to distributed modeling behaviors (Fig. 7) can be summarized as follows:

1. NC is the most influential factor due to objective model properties, such as heterogeneity owing to different implementation approaches (e.g., C++, MATLAB, LABVIEW, and DSPACE), different modeling fields (e.g., engine design, electrics,

management science, and software engineering), and geographically dispersed regions. Such heterogeneity increases the difficulty of model combination in current and future distributed simulations.

2. The second most influential factor is NSU (that is, poor interoperability). To some extent, we can say that NSU is actually the most influential factor because NC, which is constrained by the objective model properties, is difficult to mitigate and can be disregarded in actual operations. However, poor interoperability can be technically solved by first-class software practitioners during system design and development. Solving poor interoperability is confronted with obstacles even though many external interfaces (e.g., XML, OWL, BOM, Clib, Sockets) are provided by modeling tools and software. These obstacles result in NSU. For example, a unified specification for the programming development of interface agents is lacking, and effective communication and collaboration among developers during model development are insufficient.

3. The third important factor (ranked 3) is NMR. Generally, the domain knowledge and expertise of modeling engineers are rich. However, some practitioners do not fully understand the deviation from true physical models because of their subjective prejudice and outlook. These deficiencies result in inauthentic simulation models.

4. The factors with low impact (ranked 4 and 5) are NCE and NDU. As to event NCE, once software models are established, they are generally stable. Event NDU infrequently occurs in modeling processes because if a model cannot be incorporated into an existing system, then it should cease to exist.

Fig. 8a shows that the impact of various factors on inconsistent model states is continuously strengthened with increasing cycle number *C* of AMCE. Fig. 8a also indicates that events NC and NSU exert a greater influence than do NMR, NCE, and NDU. Events NMR and NCE have a general effect on NSU and NDU, whereas NDU imposes a weaker effect than do NCE, NMR, NSU, and NC. Fig. 8b shows the average influence sequence and rates of various factors at four different numbers of cycles, namely NC (33.7%)>NSU (26.6%)>NMR (20.2%)>NCE (14.5%)>NDU (5%), which is consistent with the results in Fig. 7.

6 Conclusions

We propose a directed model hyper-graph CMH and its dynamic evolution algorithm AMCE to solve the general consistency issue in modeling and model evolution. The proposed model aims to conduct a statistical investigation of the inconsistency factors in models used for service-oriented, compositional, distributed simulations. CMH can theoretically and formally describe the dynamic influencing mechanism of model inconsistency activities in terms of self-evolution, interoperability, compositionality, and authenticity. AMCE can obtain quantitative counting results by executing the CMH graph. These results are averaged to obtain the importance degree of these factors, in which different cycle numbers are established in AMCE.

The investigation in this paper is conducted from a macroscopic and theoretical perspective. Extensive simulations demonstrate that the model and algorithm are feasible, effective, and advanced. Studies on model consistency maintenance in new DSAs can offer theoretical and technical guidance for reducing model inconsistency activities and improving the reliability of modeling and simulation.

Recommendations for future work are four-fold: (1) More influencing factors should be considered and an in-depth analysis of the disturbance mechanism in modeling and its evolution should be conducted. (2) More studies on the probability of event occurrence should be carried out. (3) Our evaluation methods can be extended to some special distributed modeling applications other than simulation, including software business alignment, co-evolution theory, large-scale robotics, and similar embedded systems. (4) Corresponding maintenance technologies of model consistency can be designed based on our evaluation results.

References

Al-Jaroodi, J., Mohamed, N., 2012. Service-oriented middleware: a survey. *J. Network Comput. Appl.*, **35**(1):211-220. [doi:10.1016/j.jnca.2011.07.013]

Asiki, A., Doka, K., Konstantinou, I., *et al.*, 2009. A grid middleware for data management exploiting peer-to-peer techniques. *Future Gener. Comput. Syst.*, **25**(4):426-435. [doi:10.1016/j.future.2008.09.005]

Atzori, L., Iera, A., Morabito, G., 2010. The Internet of Things: a survey. *Comput. Networks*, **54**(15):2787-2805. [doi:10.1016/j.comnet.2010.05.010]

Beimel, D., Galanti, L., 2007. A consistency-preserving protocol for distributed collaborative model authoring. Proc. Int. Conf. on Systems Engineering and Modeling, p.118-126. [doi:10.1109/ICSEM.2007.373341]

Brandt, C., Hermann, F., Engel, T., 2009. Security and consistency of it and business models at Credit Suisse realized by graph constraints, transformation and integration using algebraic graph theory. Proc. 10th Int. Workshop on Business Process Modeling, Development and Support, p.339-352. [doi:10.1007/978-3-642-01862-6_28]

Ceranowicz, A., Cutts, D.E., Graff, J., *et al*., 2012. A proposal for a data exchange model representation standard. Proc. Spring Simulation Interoperability Workshop, p.143-155.

Chen, J.W., 2011. Comparison analysis for validating methods of system simulation models. Proc. Int. Conf. on Electric Information and Control Engineering, p.232-235. [doi:10.1109/ICEICE.2011.5778078]

Cheun, D.W., La, H.J., Kim, S.D., 2012. A taxonomic framework for autonomous service management in service-oriented architecture. *J. Zhejiang Univ.-Sci. C (Comput. & Electron.)*, **13**(5):339-354. [doi:10.1631/jzus.C1100359]

Cicirelli, F., Furfaro, A., Nigro, L., 2011. Modelling and simulation of complex manufacturing systems using state-chart-based actors. *IEEE Trans. Softw. Eng.*, **19**(2):685-703. [doi:10.1016/j.simpat.2010.10.010]

Dam, H.K., Ghose, A., 2011. An agent-based framework for distributed collaborative model evolution. Proc. 12th Int. Workshop on Principles on Software Evolution, p.121-130. [doi:10.1145/2024445.2024468]

Ding, J., Chen, L., Zhou, F., *et al*., 2005. Consistency analysis of complex declarative simulation models. *J. Softw.*, **16**(11):1868-1875 (in Chinese). [doi:10.1360/jos161868]

Engels, G., Kuster, J.M., Heckel, R., *et al*., 2002. Towards consistency-preserving model evolution. Proc. 5th Int. Workshop on Principles of Software Evolution, p.129-132. [doi:10.1145/512035.512066]

Fan, L., Ling, Y., Wang, T., *et al*., 2013. Novel clock synchronization algorithm of parametric difference for parallel and distributed simulations. *Comput. Networks*, **57**(6):1474-1487. [doi:10.1016/j.comnet.2013.02.004]

Foster, H., 2003. Semantic inconsistency and its effect on simulation. *IEE Electron. Syst. Softw.*, **1**(2):18-21. [doi:10.1049/ess:20030209]

Guo, J., Wang, Y., Trinidad, P., *et al*., 2012. Consistency maintenance for evolving feature models. *Expert Syst. Appl.*, **39**(5):4987-4998. [doi:10.1016/j.eswa.2011.10.014]

Hermann, F., Ehrig, H., Orejas, F., *et al*., 2011. Correctness of model synchronization based on triple graph grammars. Proc. 14th Int. Conf. on Model Driven Engineering Languages and Systems, p.668-682. [doi:10.1007/978-3-642-24485-8_49]

Law, A.M., 2005. How to build valid and credible simulation models. Proc. Winter Simulation Conf. [doi:10.1109/WSC.2005.1574236]

Lu, Y., Hao, Z.X., Dai, R., 2007. A sufficient and necessary condition for the absolute consistency of XML DTDs. Proc. 8th ACIS Int. Conf. on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, p.249-254. [doi:10.1109/SNPD.2007.104]

Mens, T., Wermelinger, M., Ducasse, S., *et al*., 2005. Challenges in software evolution. Proc. 8th Int. Workshop on Principles of Software Evolution, p.13-22. [doi:10.1109/IWPSE.2005.7]

Morse, K.L., 2005. XMSF profile study group final report. Technical Report No. 05F-SIW-013. Science Applications International Corporation, San Diego, CA, United States.

Singh, H.L., Gracanin, D., 2012. An approach to distributed virtual environment performance modeling: addressing system complexity and user behavior. Proc. 19th IEEE Virtual Reality Conf., p.71-72. [doi:10.1109/VR.2012.6180887]

Strassburger, S., Schulze, T., Fujimoto, R.M., 2008. Future trends in distributed simulation and distributed virtual environments: results of a peer study. Proc. Winter Simulation Conf., p.777-785. [doi:10.1109/WSC.2008.4736140]

Swinerd, C., McNaught, K.R., 2012. Design classes for hybrid simulations involving agent-based and system dynamics models. *Simul. Model. Pract. Theory*, **25**:118-133. [doi:10.1016/j.simpat.2011.09.002]

Tolk, A., Turnitsa, C.D., Diallo, S.Y., 2006. Ontological implications of the levels of conceptual interoperability model. Proc. 10th World Multi-conf. on Systemics, Cybernetics and Informatics, p.105-111.

Trollmann, F., Albayrak, S., 2011. Expressing model relations as basis for structural consistency analysis in Models@Run.Time. Proc. ACM/IEEE 15th Int. Conf. on Model Driven Engineering Languages and Systems, p.74-75. [doi:10.1145/2422518.2422530]

Uhlig, A., Rude, E., 2009. Semi automatic reliability analysis based on simulation models. Proc. Int. Conf. on Computer Applications in Shipbuilding, p.78-116.

Wang, J.Z., Varman, P., Xie, C.S., 2011. Optimizing storage performance in public cloud platforms. *J. Zhejiang Univ.-Sci. C (Comput. & Electron.)*, **12**(12):951-964. [doi:10.1631/jzus.C1100097]

Wei, O., Gurfinkel, A., Chechik, M., 2011. On the consistency, expressiveness, and precision of partial modeling formalisms. *Inf. Comput.*, **209**(1):20-47. [doi:10.1016/j.ic.2010.08.001]

Weidlich, M., Mendling, J., Weske, M., 2011. Efficient consistency measurement based on behavioral profiles of process models. *IEEE Trans. Softw. Eng.*, **37**(3):410-429. [doi:10.1109/TSE.2010.96]

Wu, Z.H., Chen, H.J., 2012. From semantic grid to knowledge service cloud. *J. Zhejiang Univ.-Sci. C (Comput. & Electron.)*, **13**(4):253-256. [doi:10.1631/jzus.C1101006]

Yan, J., 2005. Consistency and interoperability checking for component interaction rules. Proc. 12th Asia-Pacific

Software Engineering Conf., p.595-602. [doi:10.1109/APSEC.2005.55]

Yu, H., Vahdat, A., 2002. Design and evaluation of a conit-based continuous consistency model for replicated services. *ACM Trans. Comput. Syst.*, **20**(3):239-282. [doi:10.1145/566340.566342]

Yu, P., Ma, X., Cao, J., *et al*., 2013. Application mobility in pervasive computing: a survey. *Perv. Mob. Comput.*, **9**(1):2-17. [doi:10.1016/j.pmcj.2012.07.009]

Zhang, X., Ward, T., McLoone, S., 2009. Exploring an information framework for consistency maintenance in distributed interactive applications. Proc. 13th IEEE/ACM Symp. on Distributed Simulation and Real-Time Applications, p.121-128. [doi:10.1109/DS-RT.2009.23]

Zhong, W., Huang, K., 2012. Research on OWL-based BOM ontology model. Proc. 2nd Int. Conf. on Computer Application and System Modeling, p.1189-1193.

Zhou, D.X., Zhong, H., Deng, R., *et al*., 2007. Survey of simulation composability of complex systems. *J. Syst. Simul.*, **19**(8):1819-1823 (in Chinese).

Zhou, S.P., Cai, W.T., Lee, B.S., *et al*., 2004. Time-space consistency in large-scale distributed virtual environments. *ACM Trans. Model Comput. Simul.*, **14**(1):31-47. [doi:10.1145/974734.974736]

Recommended paper related to this topic

A taxonomic framework for autonomous service management in Service-Oriented Architecture

Authors: Du Wan Cheun, Hyun Jung La, Soo Dong Kim

doi:10.1631/jzus.C1100359

Journal of Zhejiang University-SCIENCE C (Computers & Electronics), 2012 Vol.13 No.5 P.339-354

Abstract: Since Service-Oriented Architecture (SOA) reveals the black box nature of services, heterogeneity, service dynamism, and service evolvability, managing services is known to be a challenging problem. Autonomic computing (AC) is a way of designing systems that can manage themselves without direct human intervention. Hence, applying the key disciplines of AC to service management is appealing. A key task of service management is to identify probable causes for symptoms detected and to devise actuation methods that can remedy the causes. In SOA, there are a number of target elements for service remedies, and there can be a number of causes associated with each target element. However, there is not yet a comprehensive taxonomy of causes that is widely accepted. The lack of cause taxonomy results in the limited possibility of remedying the problems in an autonomic way. In this paper, we first present a meta-model, extract all target elements for service fault management, and present a computing model for autonomously managing service faults. Then we define fault taxonomy for each target element and inter-relationships among the elements. Finally, we show prototype implementation using cause taxonomy and conduct experiments with the prototype for validating its applicability and effectiveness.