AMSE JOURNALS-AMSE IIETA publication-2017-Series: Advances C; Vol. 72; N°1; pp 67-85
Submitted Jan. 2017; Revised March 15, 2017, Accepted April 15, 2017

Trustworthiness Evaluation of Component-Based Software Based on Level

*Junfeng Tian, *† Zhen Li, *Zhuo Chang, **Peng Lin

* School of Computer Science and Technology, Hebei University, Baoding, China

** Department of Personnel, Hebei University, Baoding, China

(† Corresponding author: lizhen_hbu@126.com)

Abstract

The traditional trustworthiness evaluation of running software is inaccurate and incomplete because each component is as an independent unit during reliability evaluation. According to the drawbacks of traditional models, this paper proposed a trustworthiness evaluation approach of component-based software based on level. The trustworthy behavior trace diagram of component-based software was presented based on the call relation between components, and it described the trustworthy behavior trace of component in a recursive manner. With the combination of reliability and security of component, the paper proposed the dynamic trustworthiness evaluation of running software based on level and the trustworthiness evaluation approach of software based on running paths. Experiments and analyses showed that the approach could evaluate the dynamic trustworthiness of running component-based software more accurately, and could realize the trustworthiness evaluation of component-based software through the tests of running paths.

Key words

Component-based software, trustworthiness evaluation, level, reliability, security.

1. Introduction

With the rapid development of component technology, the approach of aggregating components into complex software systems is becoming mature. Component-based software engineering, constructing the software system through assembling reusable and plug-pull components, becomes the mainstream of large and complex software development paradigms. It is an appealing approach for software engineering development and industry-scale software construction. With continuous deepening of the application of component-based software in the

sensitive industries such as banking and e-commerce [1], the trustworthiness requirement of component-based software trustworthiness becomes more urgent.

If the software behavior is always accordant with the expected behavior, we call the software is trustworthy [2]. The idea of trusted computing is to perfect the terminal computer fundamentally, while the present trusted computer can only guarantee the static security of system resources. How to guarantee and measure the dynamic trustworthiness during component-based software running has become a key problem for system trustworthiness. The research on dynamic trustworthiness during component-based software running and trustworthiness evaluation of component-based software has very important significance.

For the trustworthiness of component-based software, Wen et al. [3] introduced aspect-oriented architectural design approach and relevant techniques into the design and analysis of software and offered an effective approach of software architectural design for trusted software based on monitoring. Wang et al. [4] presented a verification model for trustworthiness of interaction between software components by combining the Unified Modeling Language (UML) with Pi-calculus. Luo et al. [5] considered the dynamic change software level attributes and proposed a dynamic software reliability assessment model based on Markov chain. Mohammad et al. [6] introduced a new process for a rigorous component-centered development of trustworthy systems. Si et al. [7] presented an evaluation model for dependability of Internet-scale software on basis of Bayesian networks and trustworthiness. Elshaafi et al. [8] presented a collaborative trustworthiness determination approach using optimisation that could provide a solution to selecting trustworthy component service constructs. Chen et al. [9] proposed an interaction based requirements monitoring approach for Internetware. They collected the Internetware system behaviors in terms of actual interactions between the Internetware system and its environment, and compared the Internetware system behaviors with its specification.

Trustworthiness is a composite concept and the properties [10] contributing to it are correctness, reliability, security, availability, efficiency, etc. Among them, reliability and security are two of the most important properties, so trustworthiness can be denoted by trustworthiness ≈ reliability + security [11]. Therefore, many scholars carry out research on the reliability of component-based software. Because the path-based reliability analysis methods can be used to evaluate both the component-based software reliability and the path reliability of running component-based software, many scholars carry out research on the path-based software reliability model. Mao et al. [12] presented a general model for component-based software reliability – component probability transition diagram – based on function abstractions in order to enable reliability tracing through a dynamic process. Zhang et al. [13] introduced the dynamic transition

graph to build the relationship between the route of component-based software and component reliability, and proposed an improved component-based software reliability model based on route. Nautiyal et al. [14] presented an innovative reliability model in terms of multiple execution paths and the usage percentage of each and every component. Hsu et al. [15] proposed an adaptive framework of incorporating path testing into reliability estimation for complex component-based software system.

When the software runs to a component, the component's trustworthiness is related to its running background, internal execution path and called components. However, these models don't consider the internal execution paths and the security of components, and just consider each component as an independent unit during reliability evaluation. So the dynamic trustworthiness evaluation of software is inaccurate and incomplete. Then the accuracy of trustworthiness evaluation of component-based software based on running paths is influenced. In this paper, we make the following contributions:

First, the conception of component level is introduced considering the call relation among components in component-based software.

Second, the trustworthy behavior trace of component is described in a recursive manner, and on this basis, the trustworthy behavior trace diagram of component-based software is presented.

Third, we combine the reliability and security of component and propose the dynamic trustworthiness evaluation of running component-based software based on level and the trustworthiness evaluation approach of component-based software based on running paths.

The remainder of this paper is structured as follows: We introduce the trustworthy behavior trace of component-based software in Section 2 and discuss the simplification of trustworthy behavior trace diagram of component-based software based on association relations in Section 3. We describe the trustworthiness evaluation of component-based software in Section 4 and discuss the experimental results and analyses in Section 5. Section 6 concludes the paper.

2. Trustworthy Behavior Trace of Component-Based Software

Definition 1 (Component level). The component invoked by component-based software directly is called the first level component. The component invoked by the nth ($1 \le n \le N$) level component directly is called the (n+1)th level component. N is the maximum component layer for the component-based software.

In this paper, we assume that the transitions between the same level components obey Markov process, that is to say, the execution of next component only depends on the current component.

Definition 2 (Component). The *n*th ($1 \le n \le N$) level component $c_i(n)$ of component-based software S is denoted by $c_i(n) =< scene, G_{c_i}(n) >$.

1) *scene* is the scene of component, that is the background information of component when it begins to run. The scene of component is mainly used for security evaluation and it includes deterministic attributes and fuzzy attributes. For deterministic attributes, once any of them deviates from the normal value, the component is determined to be insecurity directly. These attributes include concurrent component, mutually exclusive component, context (call stack information), argument policies, etc. Fuzzy attributes cannot be expressed as accurate numbers. They are fuzzy and granted the prescribed error bounds. These attributes include timestamp, CPU occupancy rate, memory occupancy rate, etc.

2) $G_{c_i}(n)$ is the trustworthy behavior trace diagram of component $c_i(n)$.

Definition 3 (Meta component). If component $c_i(n)$ doesn't invoke other components, then $c_i(n)$ is called meta component.

According to the view of "trustworthiness ≈ reliability + security" [9], the trustworthiness $d_{c_i}(n)$ of meta component $c_i(n)$ can be denoted by

$$d_{c_i}(n) = (1-\omega_{c_i}) \times r_{c_i} + \omega_{c_i} \times s_{c_i}(n). \quad (1)$$

r_{c_i} is the reliability of meta component $c_i(n)$. In this paper, we assume that all meta components are the third party components. The reliability of meta components of independent development can be evaluated by the test results of them. Therefore, their reliability is unrelated to the component level. $s_{c_v}(n)$ is the security of component $c_i(n)$ and can be obtained by monitoring the scene of $c_i(n)$. For the details, see Section 4.1. ω_{c_i} is the weight of security for trustworthiness evaluation and it depends on the degree of practical requirements. If the security of meta component, such as access control, consistency verification, integrity, security protection, etc., is in high demand, the value in (0.5,1] can be assigned to ω_{c_i}. If the reliability of meta component, such as tolerance, effectiveness, recovery, etc., is in high demand, the value in [0,0.5) can be assigned to ω_{c_i}. If the security and reliability of meta component have the same degree of practical requirements, $\omega_{c_i} = 0.5$.

Definition 4 (Component-based software). The component-based software S is composed of components and the relations between two components, which is denoted by $S =< C, R >$. C is the set of the *N*th level components, denoted by $C = \{C(n) \mid n = 1, 2, ..., N\}$ where

$C(n)=\{c_i(n)|i=1,2,...,m_n\}$. N is the maximum component layer of S. m_n is the number of nth level components. R is the set of relations between the same level components.

The relations between two components with the same level include:

1) Independence relation. The components are mutually independent.

2) Association relation. The association relations between two components with the same level include:

① Sequence relation. At time t only the component $c_i(n)$ is running. When component $c_i(n)$ finishes, a subsequent component begins to run. The relationship between these two components is called sequence relation. There may be one or more than one subsequent components for selection.

② Concurrent relation. In concurrent environment, if two or more components are running at the same time or overlapped running, the relationship between any two components is called concurrent relation. The concurrent component, as an attribute of scene, can be used to evaluate the security of component.

③ Fault tolerance relation. A group of components with fault tolerance relation is composed of a primary component and a group of backup components. The primary component and backup components can be released by several program solutions. When the primary component is failed, the first backup component becomes the new primary component; when the new primary component is failed, the second backup component becomes the new primary component; and so on. When all fault tolerance components are failed, the software is failed. The association relation can be used to simplify the trustworthy behavior trace diagram of component-based software. See Section 3 for details.

3) Exclusive relation. If component $c_i(n)$ and component $c_j(n)$ $(i \neq j)$ cannot run at the same time, the relationship between these two components is called exclusive relation. The components with exclusive relation are as follows: components in two or more branch paths, components contending for critical resource, component with interruption and component in the interrupt handling routine, and so on. The mutually exclusive component, as an attribute of scene, can be used to evaluate the security of component.

Definition 5 (Trustworthy behavior trace diagram of component). The trustworthy behavior trace diagram $G_{c_i}(n)$ of component $c_i(n)$ is denoted by

$$G_{c_i}(n)=\begin{cases} < c_i(n), d_{c_i}(n) >, & c_i(n) \text{ is a meta component;} \\ < G_{C_i(n+1)}(n+1), T(n+1), s(n+1), & \\ \quad e(n+1), P(n+1), D(n+1) >, & \text{else.} \end{cases} \tag{2}$$

1) $d_{c_i}(n)$ is the trustworthiness of component $c_i(n)$.

2) $G_{C_i(n+1)}(n+1)=\{G_{c_j}(n+1)\,|\,c_j(n+1)\in C_i(n+1)\}$, where $C_i(n+1)$ is the set of the (n+1)th level components for component $c_i(n)$.

3) $T(n+1)$ is the transition set of the (n+1)th level components. The transition of component $c_\alpha(n+1)$ to component $c_\beta(n+1)$ is denoted by $(c_\alpha(n+1),c_\beta(n+1))$.

4) $s(n+1)$ is the start component, $s(n+1)\in C(n+1)$.

5) $e(n+1)$ is the end component, $e(n+1)\in C(n+1)$.

6) $P(n+1)$ is the universe of transition probabilities for the (n+1)th level components, that is $\{x\,|\,0\le x\le 1\}$. The transition probabilities between component $c_\alpha(n+1)$ and component $c_\beta(n+1)$ is denoted by $p_{\alpha,\beta}(n+1\,|\,c_i)$ which can be abbreviated as $p_{\alpha,\beta}$ if the component level and caller component are not concerned. $c_i(n)$ is the caller component of $c_\alpha(n+1)$ and $c_\beta(n+1)$.

7) $D(n+1)$ is the trustworthiness universe of the (n+1)th level components, that is $\{x\,|\,0\le x\le 1\}$. $d_{c_\alpha}(n+1)$ is the trustworthiness of component $c_\alpha(n+1)$ which can be abbreviated as d_{c_α} if the component level is not concerned.

Definition 6 (Trustworthy behavior trace diagram of component-based software). The trustworthy behavior of component-based software S is denoted by trustworthy behavior trace diagram of component-based software $G=<C(1),T,s,e,P,D>$.

1) $C(1)$ is the set of the first level components.

2) T is the transition set of the first level components. The transition of component $c_\alpha(1)$ to component $c_\beta(1)$ is denoted by $(c_\alpha(1),c_\beta(1))$.

3) s is the start component, $s\in C(1)$.

4) e is the end component, $e\in C(1)$.

5) P is the universe of transition probabilities for the first level components, that is $\{x\,|\,0\le x\le 1\}$. The transition probabilities between component $c_\alpha(1)$ and component $c_\beta(1)$ is denoted by $p_{\alpha,\beta}(1)$ abbreviated as $p_{\alpha,\beta}$.

6) D is the trustworthiness universe of the first level components, that is $\{x\,|\,0\le x\le 1\}$. $d_{c_\alpha}(1)$ is the trustworthiness of component $c_\alpha(1)$ abbreviated as d_{c_α}.

In the paper, we only consider one single input component and one single output component. The input component is the start component and the output component is the end component. For

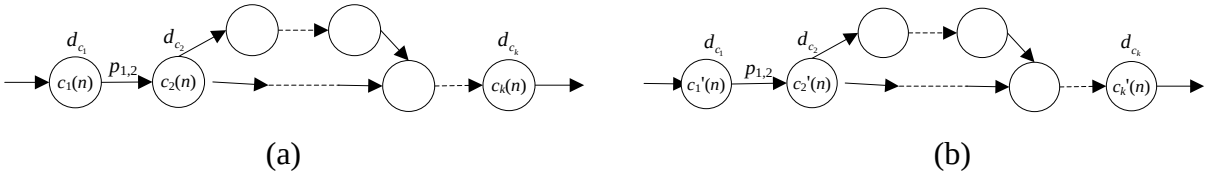
multiple input components and multiple output components, we can revise it to one single input component and one single output component by introducing a start component connecting multiple input components and an end component connecting multiple output components.

3. Simplification of Trustworthy Behavior Trace Diagram of Component-Based Software

Because the structure and the function of component-based software are complex and the relations among the same level components are multiple, the trustworthy behavior trace diagram of component-based software is large, which is not satisfied with the need of efficient evaluation. Therefore, the trustworthy behavior trace diagram of component-based software needs to be simplified. Based on the different type of association relations between components in $C(n)$, the trustworthy behavior trace diagram of component-based software is simplified so that the association relation between two components with the same level only have sequence relation after simplification.

1) Sequence style

The components of this type execute in sequence, as shown in Figure 1. The component structure of this style does not need simplification. The component $c_i(n)$ before simplification is corresponding to the component $c_i'(n)$ after simplification and their trustworthiness and transition probabilities are all the same.



(a) (b)

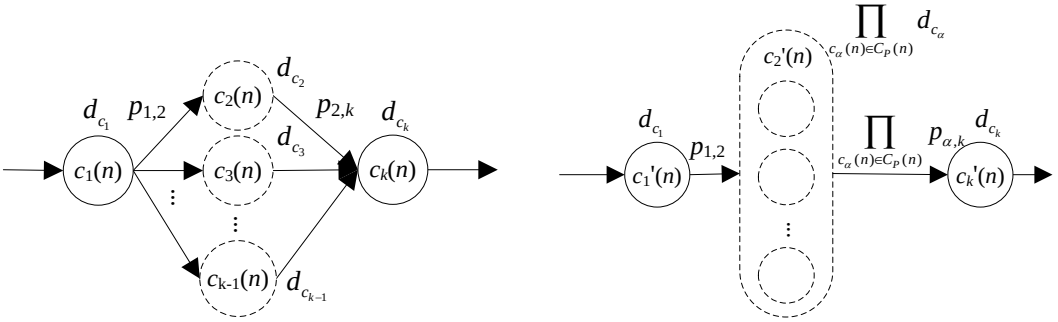
Fig. 1. Sequence style (a) Before simplification (b) After simplification

2) Concurrent style

In concurrent environment, the performance of system can be improved by running multiple components concurrently. Because components running concurrently in the trustworthy behavior trace diagram of component-based software have the same scene, front component and next component, the components running concurrently can be merged into one component for trustworthiness evaluation.

In Figure 2(a), the dotted line denotes part or all of component $c_2(n)$, …, $c_{k-1}(n)$ run concurrently. All the transition probabilities between component $c_1(n)$ and component $c_2(n), c_3(n), \ldots, c_{k-1}(n)$ are $p_{1,2}$. Let the set of components running concurrently in a scene be $C_P(n)$ which is a subset of $\{ c_2(n), c_3(n), \ldots, c_{k-1}(n) \}$. When the components in $C_P(n)$ run concurrently, the components in $C_P(n)$ can be as a whole and the trustworthiness to outside is $\prod_{c_\alpha(n) \in C_P(n)} d_{c_\alpha}$.

The component $c_1(n)$ and $c_k(n)$ before simplification are corresponding to the component $c_1'(n)$ and $c_k'(n)$ after simplification respectively. The components in $C_P(n)$ are merged into one component $c_2'(n)$. The trustworthiness of $c_1'(n)$ and $c_k'(n)$ are the trustworthiness of $c_1(n)$ and $c_k(n)$ respectively, and the trustworthiness of $c_2'(n)$ is $\prod_{c_\alpha(n) \in C_P(n)} d_{c_\alpha}$. The transition probability between component $c_1'(n)$ and component $c_2'(n)$ is $p_{1,2}$, and the transition probability between component $c_2'(n)$ and component $c_k'(n)$ is the product of the transition probabilities between each component in $C_P(n)$ and component $c_k(n)$, that is $\prod_{c_\alpha(n) \in C_P(n)} p_{\alpha,k}$, as is shown in Figure 2(b).



(a) (b)

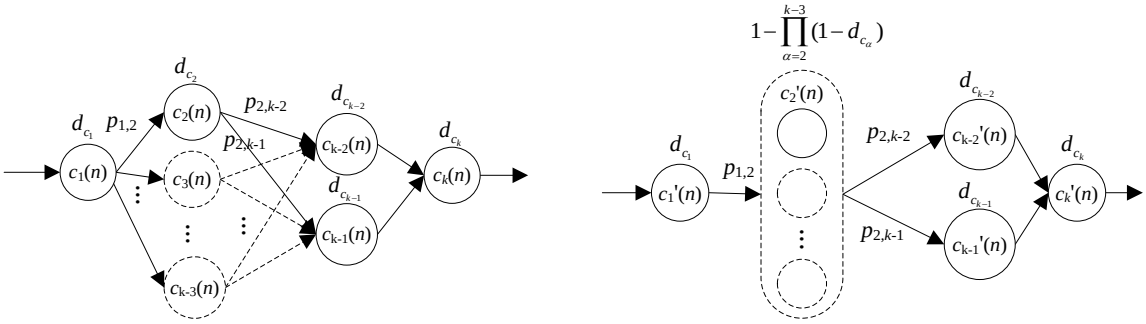
Fig. 2. Concurrent style (a) Before simplification (b) After simplification

3) Fault tolerance style

Only one of components with fault tolerance relation runs at a certain time, so the components with fault tolerance relation can be merged into one component.

In Figure 3(a), the component $c_3(n)$, …, $c_{k-3}(n)$ denoted by dotted line are the backup components of primary component $c_2(n)$. At a certain time, only one component

$c_\alpha(n)$ ($\alpha = 2,3,...,k-3$) runs, so component $c_2(n)$,…, $c_{k-3}(n)$ can be merged into component $c_2'(n)$. When all component $c_2(n)$,…, $c_{k-3}(n)$ are untrustworthy, component $c_2'(n)$ is untrustworthy, so the trustworthiness of component $c_2'(n)$ is $1-\prod_{\alpha=2}^{k-3}(1-d_{c_\alpha})$. The transition probability between component $c_2'(n)$ and component $c_{k-2}'(n)$ or $c_{k-1}'(n)$ is the same as that between the primary component $c_2(n)$ and component $c_{k-2}'(n)$ or $c_{k-1}'(n)$, as is shown in Figure 3(b).



(a) (b)

Fig. 3. Fault tolerance style (a) Before simplification (b) After simplification

4. Trustworthiness Evaluation of Component-Based Software

4.1. Dynamic Trustworthiness Evaluation of Running Component-Based Software

The trustworthy behavior trace diagram of component-based software G is denoted by $G' =< C'(1), T', s', e', P', D' >$ after simplification.

Definition 7 (Running path). In G', if there is a path $l: c_1'(1)c_2'(1)\cdots c_q'(1)$ that is satisfied with $c_1'(1) = s'$, the path is called a running path with final component $c_q'(1)$. If $c_q'(1) = e'$, the path is called a running path of G'.

The trustworthiness of running path $l: c_1'(1)c_2'(1)\cdots c_q'(1)$ with final component $c_q'(1)$ is

$$d_l = \prod_{c_j'(1)\in l} d_{c_j'}(1) \quad . \tag{3}$$

The trustworthiness of any of the nth level component $c_v'(n)$ for component $c_j'(1)$ in the running path l is

$$d_{c_v'}(n)=\begin{cases}\omega_{c_v'}\times s_{c_v'}(n)+(1-\omega_{c_v'})\times r_{c_v'}, & c_v'(n) \text{ is a meta component;}\\ \prod\limits_{c_u'(n+1)\in l(n+1)} d_{c_u'}(n+1), & \text{else.}\end{cases}$$

(4)

where $n\geq 1$, $l(n+1)$ is the running path of l in $G_{c_v'}(n)$, $s_{c_v'}(n)$ is the security of component $c_v'(n)$, and $\omega_{c_v'}$ is the weight of security for trustworthiness evaluation. The value of $\omega_{c_v'}$ depends on the practical requirements of reliability and security and the details are in Section 2.

$s_{c_v'}(n)$ can be achieved based on the scene of component $c_v'(n)$. If the components with concurrent relation don't run concurrently, or the components with exclusive relation run at the same time, or the context or argument deviates from the normal, then $s_{c_v'}(n)=0$; otherwise, $s_{c_v'}(n)$ depends on the fuzzy attributes and $s_{c_v'}(n)\in[0,1]$. The evaluation process is as follows:

In the training phase, for component $c_v'(n)$, the training sample X_i $(1\leq i\leq t)$ ($X_i=[x_{i1},x_{i2},...,x_{ik}]$) of fuzzy attributes is captured. Let $x_{1j},x_{2j},...,x_{tj}$ be t sample values of the fuzzy attribute A_j. After removing various effects of the environment, fuzzy attributes are approximately normally distributed and the normal value of each fuzzy attribute fluctuates around the average value. Therefore, we can determine the security of fuzzy attribute according to the degree of deviation from the average value. The average value μ_j of fuzzy attribute A_j is as follows:

$$\mu_j=(\sum_{i=1}^{t}x_{ij})/t \quad .$$

(5)

The test sample $Y=[y_1,y_2,...,y_k]$ is captured during the component-based software actual running. The security $s_{c_v'}(n)_j$ of fuzzy attribute A_j for sample Y is as follows:

$$s_{c_v'}(n)_j=\begin{cases}\dfrac{y_j}{\mu_j}, & y_j\in[x\min_j,\mu_j]\\ 1+\dfrac{\mu_j}{x\max_j}-\dfrac{y_j}{x\max_j}, & y_j\in[\mu_j,x\max_j]\\ 0, & \text{else}\end{cases} ,$$

(6)

where $x\min_j=\min\limits_{1\leq i\leq t}\{x_{ij}\}$, $x\max_j=\max\limits_{1\leq i\leq t}\{x_{ij}\}$, $j=1,2,...,k$.

The security $s_{c_v'}(n)$ of sample Y is $s_{c_v'}(n)=\sum_{j=1}^{k} w_j s_{c_v'}(n)_j$, where w_j is the weight of fuzzy attribute A_j for component $c_v'(n)$ and can be determined subjectively or objectively.

The threshold τ $(\tau \in (0,1))$ of software trustworthiness can be set through experiments involving a large number of running paths with known trustworthiness. When the trustworthiness d_l of l is less than the threshold τ, the software is untrustworthy when it runs to component $c_q'(1)$ and should be stop running.

Through the experiments involving a large number of running traces with known trustworthiness, we found that the detection accuracy is approximately normally distributed when the threshold τ of software trustworthiness changes from 0 to 1. When τ is close to τ_0, the detection accuracy is highest. The bigger the value of τ is, the higher the false positive rate is; the smaller the value of τ is, the higher the false negative rate is. Then the threshold $\tau=\tau_0$.

4.2. Trustworthiness Evaluation of Component-Based Software Based on Running Paths

The probability of running path $l: c_1'(1)c_2'(1)\cdots c_q'(1)$ ($c_1'(1)=s'$, $c_q'(1)=e'$) in G' is

$$f_l=\begin{cases}\prod\limits_{c_j'(1)\in l \wedge c_j'(1)\neq e'} p_{j',(j+1)'}(1), & c_1'(1)\neq c_q'(1);\\ 1, & c_1'(1)=c_q'(1).\end{cases} \tag{7}$$

The trustworthiness of running path l in G' is

$$d_l=\prod_{c_j'(1)\in l} d_{c_j'}(1) \tag{8}$$

.

The trustworthiness of the *n*th level component $c_v'(n)$ for component $c_j'(1)$ is

$$d_{c_v'}(n)=\begin{cases}\omega\times s_{c_v'}(n)+(1-\omega)\times r_{c_v'}, & c_v'(n) \text{ is a meta component;}\\ d_{s'(n+1)}(n+1), & s'(n+1)=e'(n+1);\\ \dfrac{\sum\limits_{l(n+1)\in L_{c_v'}(n+1)}\left(\prod\limits_{c_u'(n+1)\in l(n+1)} d_{c_u'}(n+1)\times \prod\limits_{\substack{c_u'(n+1)\in l(n+1)\\ \wedge c_u'(n+1)\neq e'(n+1)}} p_{u',(u+1)'}(n+1)\right)}{\sum\limits_{l(n+1)\in L_{c_v'}(n+1)}\left(\prod\limits_{\substack{c_u'(n+1)\in l(n+1)\\ \wedge c_u'(n+1)\neq e'(n+1)}} p_{u',(u+1)'}(n+1)\right)}, & \text{else,}\end{cases}$$

(9)

where $n\geq 1$, $l(n+1)$ is the running path of l in $G_{c_v'}'(n)$, $L_{c_v'}(n+1)$ is the set of running paths in $G_{c_v'}'(n)$ for test. The calculation process of $s_{c_v'}(n)$ can be found in Section 4.1.

The trustworthiness of component-based software for G' is:

$$d=\frac{\sum\limits_{l\in L}(d_l\times f_l)}{\sum\limits_{l\in L} f_l},$$

(10)

where L is the set of running paths in G' for test. The wider the range of running paths for test is, the more accurate the trustworthiness of component-based software we can get is.

5. Experiments and Analyses

Our experiment was carried on taking a simulator of an ATM bank system [16] for example. The software, often used in software reliability or trustworthiness analysis, consisted of eleven components and contained six natural faults. In order to have a better representation of call relation between components, we extended the original software and added seven new components shown in the dashed line frame of Figure 4. According to the two running paths in the extended software, we evaluated their dynamic trustworthiness for normal running trace and abnormal running trace respectively and discussed the threshold selection. We also evaluated the trustworthiness of software by testing running paths in the software, and verified the effectiveness of our approach.

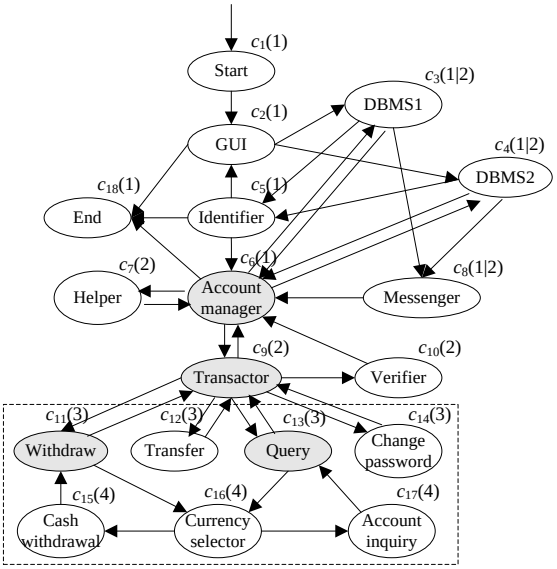


Fig. 4. The extended ATM bank system architecture

There are eight meta components containing eight natural faults altogether in the extended software. These meta components are $c_2, c_3, c_4, c_7, c_{10}, c_{14}, c_{15}, c_{17}$. In Figure 4, the components with shadow are caller components and the components without shadow are meta components. The numbers in bracket after component name are the component level. Some components such as c_3, c_4, c_8 has multiple levels considering several running paths. Their different levels are separated by "|".

According to these eight faults, nine versions of the system were constructed, in which version 9 contained all the faults in component c_7, c_{10}, c_{14}, c_{15} and c_{17} but the others contained one fault each. We randomly generated inputs to estimate the reliability of each individual faulty meta component until it was converged. Similarly, to compute the reliability of the caller component c_6, version 9 was used. The operational behaviors in this experiment were collected to calculate the transition probability between components. The reliability of meta-components are as follows: r_{c_1} =1.0, r_{c_2} =0.98, r_{c_3} =0.97, r_{c_4} =0.97, r_{c_5} =1.0, r_{c_7} =0.99, r_{c_8} =1.0, $r_{c_{10}}$ =0.98, $r_{c_{12}}$ =1.0, $r_{c_{14}}$ =0.99, $r_{c_{15}}$ =0.99, $r_{c_{16}}$ =1.0, $r_{c_{17}}$ =0.98, $r_{c_{18}}$ =1.0. The reliability of caller component c_6 is $r_{c_6}(1)$ =0.95. The transition probability between components is shown in Table 1.

Table 1. The transition probability between components

omponent	The transition probability
c_1	$p_{1,2}(1) = 1.0$
c_2	$p_{2,3}(1) = p_{2,4}(1) = 0.999$, $p_{2,18}(1) = 0.001$

c_3	$p_{3,5}(1)=1.0$, $p_{3,6}(2\mid c_6)=0.669$, $p_{3,8}(2\mid c_6)=0.331$
c_4	$p_{4,5}(1)=1.0$, $p_{4,6}(2\mid c_6)=0.669$, $p_{4,8}(2\mid c_6)=0.331$
c_5	$p_{5,2}(1)=0.048$, $p_{5,6}(1)=0.951$, $p_{5,18}(1)=0.001$
c_6	$p_{6,3}(2\mid c_6)=0.4239$, $p_{6,4}(2\mid c_6)=0.4239$, $p_{6,7}(2\mid c_6)=0.1612$, $p_{6,9}(2\mid c_6)=0.4149$, $p_{6,18}(1)=1.0$
c_7	$p_{7,6}(2\mid c_6)=1.0$
c_8	$p_{8,6}(2\mid c_6)=1.0$
c_9	$p_{9,6}(2\mid c_6)=0.01$, $p_{9,10}(2\mid c_6)=0.99$, $p_{9,11}(3\mid c_9)=0.425$, $p_{9,12}(3\mid c_9)=0.161$, $p_{9,13}(3\mid c_9)=0.31$, $p_{9,14}(3\mid c_9)=0.1$
c_{10}	$p_{10,6}(2\mid c_6)=1.0$
c_{11}	$p_{11,9}(3\mid c_9)=1.0$, $p_{11,16}(4\mid c_{11})=1.0$
c_{12}	$p_{12,9}(3\mid c_9)=1.0$
c_{13}	$p_{13,9}(3\mid c_9)=1.0$, $p_{13,16}(4\mid c_{13})=1.0$
c_{14}	$p_{14,9}(3\mid c_9)=1.0$
c_{15}	$p_{15,11}(4\mid c_{11})=1.0$
c_{16}	$p_{16,15}(4\mid c_{11})=1.0$, $p_{16,17}(4\mid c_{17})=1.0$
c_{17}	$p_{17,13}(4\mid c_{13})=1.0$

In the extended ATM bank system, component c_4 is a backup component of component c_3 in order to improve the fault tolerant ability, that is the relationship of component c_3 and component c_4 is fault tolerant relation. Therefore, during the simplification of trustworthy behavior trace diagram of component-based software, component c_3 and component c_4 should be merged into one component c_3', and other component c_i ($1\le i\le 18$, $i\ne 3, i\ne 4$) remains unchanged and is corresponding to the component c_i'. The trustworthiness of the first level component c_3' is $d_{c_3'}(1) = 1-[1-d_{c_3}(1)]\times[1-d_{c_4}(1)]$ and the trustworthiness of the second level component c_3' is $d_{c_3'}(2)=1-[1-d_{c_3}(2)]\times[1-d_{c_4}(2)]$.

According to the running paths shown in Figure 5, we evaluate the dynamic trustworthiness of running component-based software. In Figure 5(a), the execution sequence of component in running path Pt1 is: $c_1', c_2', c_3', c_5', c_6'(c_9'(c_{11}'(c_{16}', c_{15}')), c_{10}'), c_6'(c_3', c_8')$, c_{18}'. In Figure 5(b), the execution sequence of component in running path Pt2 is: $c_1', c_2', c_3', c_5', c_6'(c_9'(c_{14}')$, $c_{10}')$, $c_6'(c_3')$, c_{18}'. Among them, the call relations are denoted by brackets. For running path Pt1 and Pt2, we discuss three running traces respectively. When the software runs along Normal trace, the

software is trustworthy; when it runs along Abnormal trace 1 or Abnormal trace 2, the software is untrustworthy.

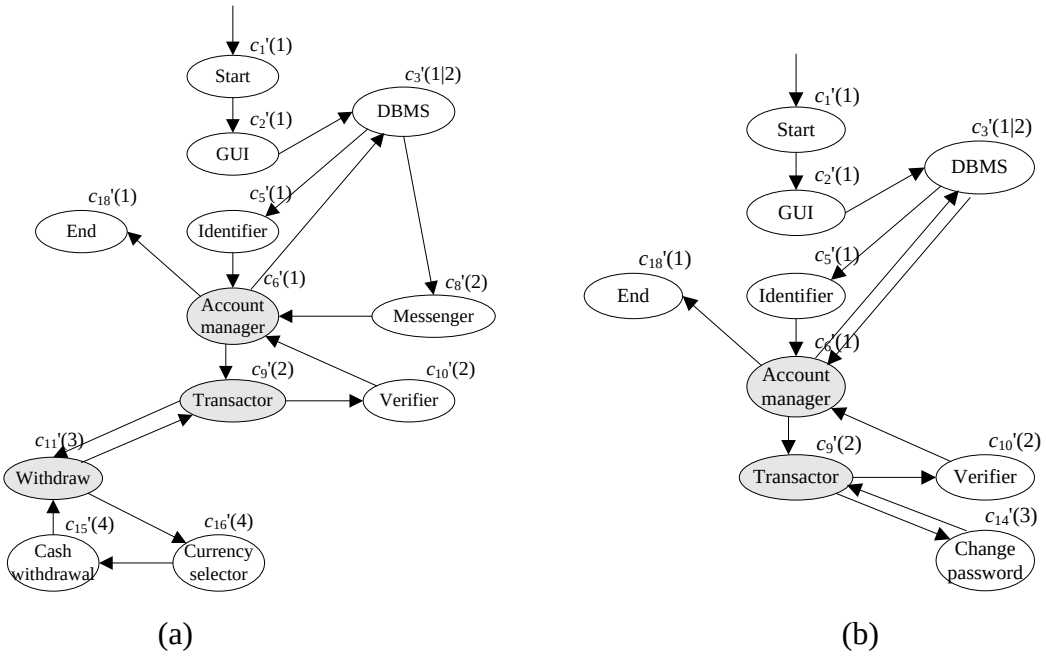


Fig. 5. Running paths (a) Pt1 (b) Pt2

1) Normal trace: A trace that the software runs normally.

2) Abnormal trace 1: The codes of changing the input arguments of component c_{10}' are added between component c_9' and component c_{10}'.

3) Abnormal trace 2: The codes that don't changing the context and argument policies are added between component c_9' and component c_{10}'.

The value of ω_{c_v} depends on the component requirements of reliability and security. According to the practical requirements of ATM bank system, component c_{10}'(Verifier) should have higher security and its security weight is set to 0.7; The security and reliability of other components have the same demand degree, so these components' security weight is set to 0.5.

The traditional approaches [12-15] do not consider the running paths and the security of caller component and each component is as an independent unit during reliability evaluation. In order to discuss the trustworthiness of running path based on reliability and security, the security of component is added to traditional approaches just considering reliability, which is denoted by "traditional approach + security". The calculation process of the security of component can be found in Section 4.1. The trustworthiness of component is the weighted average value of reliability

and security (the security weight of component $c_v'(n)$ is $\omega_{c_v'}$). The trustworthiness of each running trace for running path Pt1 and Pt2 evaluated by different approaches is shown in Table 2.

Table 2. Trustworthiness of running path Pt1 and Pt2

Running path	Running trace	Reliability of running path (traditional approach)	Trustworthiness of running path	
			Our approach	Traditional approach + security
Pt1	Normal trace	0.86	0.90	0.88
	Abnormal trace 1	0.86	0.28	0.88
	Abnormal trace 2	0.86	0.69	0.71
Pt2	Normal trace	0.88	0.92	0.90
	Abnormal trace 1	0.88	0.27	0.90
	Abnormal trace 2	0.88	0.40	0.45

Through the experiments involving a large number of running traces with known trustworthiness, we found that the detection accuracy is approximately normally distributed when the threshold τ of software trustworthiness changes from 0 to 1. When τ is close to 0.85, the detection accuracy is highest. The bigger the value of τ is, the higher the false positive rate is; the smaller the value of τ is, the higher the false negative rate is. The trustworthiness evaluation of running traces for different threshold τ is shown in Table 3. In Table 3, "T" denotes trustworthy and "U" denotes untrustworthy. τ =0.95 is taken as an example of bigger τ and τ =0.65 is taken as an example of smaller τ. When τ =0.95, the trustworthiness of Normal trace for running path Pt1 or Pt2 is determined to be untrustworthy inaccurately for two approaches; when τ =0.65, the trustworthiness of Abnormal trace 1 or Abnormal trace 2 is determined to be trustworthy inaccurately for two approaches. We mainly discuss the situation of τ =0.85 below.

Table 3. Trustworthiness evaluation of running path for different threshold τ

Running path	Running trace	Trustworthiness evaluation of running path					
		Our approach			Traditional approach + security		
		τ =0.95	τ =0.85	τ =0.65	τ =0.95	τ =0.85	τ =0.65
Pt1	Normal trace	U	T	T	U	T	T
	Abnormal trace 1	U	U	U	U	T	T
	Abnormal trace 2	U	U	T	U	U	T
Pt2	Normal trace	U	T	T	U	T	T
	Abnormal trace 1	U	U	U	U	T	T
	Abnormal trace 2	U	U	U	U	U	U

For Normal trace, the trustworthiness of each running path (Pt1 or Pt2) is greater than τ and each running path is determined to trustworthy accurately for two approaches. For Abnormal trace

1, when the software runs along each running path, our approach can detect the component c_{10}' is not satisfied with its argument policies, so $s_{c_{10}'}(2)=0$, the trustworthiness of each running path is less than τ and each running path is determined to be untrustworthy accurately; "Traditional approach + security" cannot find the abnormal of arguments passed between called components, so the trustworthiness of each running path is greater than τ and each running path is determined to be trustworthy inaccurately. For Abnormal trace 2, our approach can detect that the fuzzy attributes of scene such as timestamp deviate, so the trustworthiness of each running path is less than τ and each running path is determined to be untrustworthy accurately; "Traditional approach + security" can also determine the trustworthiness correctly, but the grain is coarse and the abnormal of component c_6' called for the first time cannot be found until calling the component c_6' for the second time with abnormal scene. Therefore, "traditional approach + security" cannot distinguish the reliabilities of component c_9' calling different components and do not consider the security of inside components called by component c_6', so that the trustworthiness of caller component c_6' is inaccurate, which influences the accuracy of trustworthiness evaluation of running path.

In the experiment, we test the trustworthiness of component-based software from three situations listed in Table 4. For test 1, the trustworthiness of component-based software for two approaches is greater than τ and the software is determined to be trustworthy accurately. For test 2, the trustworthiness of component-based software for two approaches is less than τ and the software is determined to be untrustworthy accurately. For test 3, our approach can determine the trustworthiness of component-based software accurately, while because "traditional approach + security" cannot find the abnormal of arguments passed between called components, each trace with abnormal component arguments is determined to be trustworthy inaccurately, the trustworthiness of component-based software is greater than τ and the software is determined to be trustworthy inaccurately. Therefore, our approach can evaluate the trustworthiness of component-based software more accurately.

Table 4. Trustworthiness of component-based software

Test No.	Running traces for test	Trustworthiness of component-based software	
		Our approach	Traditional approach + security
1	15 normal traces	0.91	0.92
2	5 traces with abnormal fuzzy attributes in scene +10 normal traces	0.63	0.67
3	5 traces with abnormal component arguments +10 normal traces	0.55	0.91

6. Conclusion

According to the inaccurate and incomplete trustworthiness evaluation of running software for traditional models, the conception of component level was introduced. The reliability and security of component were combined and the trustworthiness evaluation approach of component-based software based on level was proposed. Experiments and analyses showed that our approach could evaluate the dynamic trustworthiness of component-based software and the trustworthiness of component-based software more accurately. Our future work is to study the best test range of running paths involving loop structure during the trustworthiness evaluation of component-based software so that we can achieve higher accuracy of trustworthiness of component-based software with less cost.

Acknowledgments

This paper was supported by the National Natural Science Foundation of China (61170254), the Natural Science Foundation of Hebei Province (F2015201089, F2014201099), the Youth Foundation of Hebei Educational Committee (QN2016149), and the Science Foundation of Hebei University (2013-250).

References

1. H. Pang, Research on optimization of B2C e-commerce service quality oriented by customer demand, 2016, Revista de la Facultad de Ingeniería, vol. 31, no. 6, pp.103-113.
2. TCG specification architecture overview, http://www.trustedcomputinggroup.org/tcg-architecture-overview-version-1-4, May 2016.
3. J. Wen, H.M. Wang, S. Ying, Y.C. Ni, T. Wang, Toward a software architectural design approach for trusted software based on monitoring, 2010, Chinese Journal of Computers, vol. 33, no. 12, pp.2321-2334.
4. D. Wang, J.S. Chang, W.B. Zhao, Verification model for trustworthiness of interaction between software components with Pi-calculus, 2012, Journal of Frontiers of Computer Science and Technology, vol. 6, no. 5, pp.419-429.
5. X.X. Luo, Z.Y. Tang, Y.J. Zhao, Dynamic software reliability assessment based on Markov chain, 2015, Application Research of Computers, vol. 32, no. 8, pp.2400-2405.
6. M. Mohammad, V. Alagar, A component-based development provess for trustworthy systems, 2012, Journal on Software: Evolution and Process, vol. 24, no. 7, pp.815-835.

7. G.N. Si, J. Xu, J.F. Yang, W. Shuo, An evaluation model for dependability of Internet-scale software on basis of Bayesian networks and trustworthiness, 2014, Journal of Systems and Software, vol. 89, pp.63-75.
8. H. Elshaafi, D. Botvich, Optimisation-based collaborative determination of component trustworthiness in service compositions, 2016, Security and Communication Networks, vol. 9, no. 6, pp.513-527.
9. X.H. Chen, J. Liu, Z.M. Liu. Requirements monitoring for Internetware: an interaction based approach, 2013, SCIENCE CHINA: Information Science, vol. 56, no. 8, pp.1-15.
10. X.L. Liu, A.X. Wang, L. Wang, Quality evaluation model and algorithm of software system based on fuzzy closeness degree, 2016, Revista de la Facultad de Ingeniería, vol.31, no.7, pp.144-151.
11. C.X. Shen, H.G. Zhang, H.M. Wang, J. Wang, B. Zhao, F. Yan, F.J. Yu, L.Q. Zhang, M.D. Xu, Research on trusted computing and its development, 2010, SCIENCE CHINA: Information Sciences, vol. 40, no. 2, pp.139-166.
12. X.G.Mao, Y.J. Deng, A general model for component-based software reliability, 2004, Journal of Software, vol. 15, no. 1, pp.27-32.
13. W. Zhang, W.Q. Zhang, Study of improved component-based software reliability model based on route, 2011, Computer Science, vol. 38, no. 2, pp.148-151.
14. L. Nautiyal, N. Gupta, S.C. Dimri, Measurement of the reliability of a component-based development using a path-based approach, 2014 ACM SIGSOFT Software Engineering Notes, vol. 39, no. 6, pp.1-5.
15. C.J. Hsu, C.Y. Huang, An adaptive reliability analysis using path testing for complex component-based software systems, 2011, IEEE Transactions on Reliability, vol. 60, no. 1, pp.158-170.
16. W.L.Wang, Y. Wu, M.H. Chen, An architecture-based software reliability model, 1999, Proceedings of the 5th Pacific Rim International Symposium on Dependable Computing, Hong Kong, pp. 143-150.