Research on Protocol-Level Behavioral Substitutability of Software Components in Component-based Software System

HAIYANG HU[1,2] HUA HU[1,3]
[1]College of Computer Science and Information Engineering,
Zhejiang Gongshang University, Hangzhou 310018
[2]State Key Laboratory for Novel Software Technology,
Nanjing University, Nanjing 210093
[3]Hangzhou Dianzi University, Zhejiang, China, 310018

Abstract: The component-based software development (CBSD) has been paid more attention by software practicers in recent years. How to analyze and verify behavior-level component substitutability is very important when the component-based software system needs upgrading or maintaining. Concentrating on the component-based software system, this paper formally specifies the components and their interaction behaviors, analyzes the behavior of the new component compared with the old one, and then presents a set of rules for verifying behavioral substitutability of components in software system to ensure the behavioral compatibility whenever a component is replaced by a new one. Finally, an example of e-commerce is presented to illustrate the feasibility and pertinence of this approach.

Key-Words: software component component composition behavioral compatibility behavioral substitutability

1 Introduction

Being an important direction in software engineering research [1], component-based software development (CBSD) has been paid more attention by software researchers and developers. The component-based software system has such advantages as adaptability, flexibility and easy maintenance. Moreover, component-based software system can improve development efficiency and software quality through reusing software component to construct a complex software system. Thus, it can make the software development timely to meet the changes of the market. From the 1970s to now, various component technology and products continue to emerge, and there is a large number of studies working on it. [1,2]

When we upgrade and maintain the component-based software system, we often need to take into account whether an old component can be replaced by a new one or not, and whether the behavior of the entire system after replacement can still preserve compatible. To meet this requirement, at present, several mature object-oriented component productions (such as CORBA□EJB□COM/DCOM) describe and standardize the external interaction among components through the Interface Definition Languages (IDLs). However, IDLs only defines the syntax of component interaction, such as the number of parameters, the types of parameters and their sequences in the interface. Hence, the approach of IDLs can't support ensuring the correctness of behavioral interactions among components. From the late 1990s to now, the technology of describing and verifying behavioral interactions among components has been focused on by researchers. Meanwhile, most works [3][4][5][6][7] only consider the components' substitutability under the case where the provided interfaces of new component differs from the old one's, and they rarely take into account the case where the components replaced can also have requested interfaces to the external environments at the same time. On the other hand, the components contained in a software system may be distributed in the network environment, and provided by different providers. These providers may not be able to know exactly about the specific behavioral requests from the external users to the component. Hence, how to replace a component without affecting all external users needs a further study.

In this paper, based on process algebra, we present a set of rules for verifying protocol-level behavioral substitutability of components in software system to ensure the behavioral compatibility in the updated system. The rules include: for an assembly containing only two components, 1) the rule for ensuring substitutability when a component is replaced by a new component with its provided interface expanding; 2) the rule for ensuring substitutability when the new component has changed the behavioral of both its provided and requested interfaces. Based on these rules, we present

the rules for behavioral substitutability in the multiple-component software system.

In the remainder of this paper, section 2 overviews the basic knowledge and concepts required. In section 3, we present a formal definition for a component with its interaction behavior expanded. The rules for behavioral substitutability among components are presented in section 4. In section 5, we illustrate the features of this paper by a specific e-commerce example. In section 6, we discuss the related work and give a conclusion of this paper.

2 Basic Concepts

Similar to the related work of current researchers [8][9][10], this paper formally describes the external behavior of component based on process algebra. We use PA that is proposed by Bernardo.M [11] to formally describe and verify the behavioral substitutability.

Definition 1 The process terms of PA is generated by the following syntax:

$E ::= \underline{0} \mid a.E \mid E/L \mid E\backslash L \mid E[\varphi] \mid E_1 + E_2 \mid E_1 \|_S E_2 \mid A$

- $\underline{0}$ is the term that can't perform any action.
- "$a.E$" can execute action a and then behaves as term E.
- "E/L" behaves as term E except that each executed action a is hidden, i.e. turned into τ, whenever $a \in L$.
- "$E\backslash L$" behaves as term E except that each executed action a is forbidden, whenever $a \in L$, and $E\backslash L \equiv E \|_L \underline{0}$.
- "$E[\varphi]$" behaves as terms E except that each execution a becomes $\varphi(a)$;
- "$E_1 \|_S E_2$" asynchronously executes actions of E_1 or E_2 not belonging to S and synchronously executes actions of E_1 and E_2 belonging to S.
- $E_1 + E_2$ behaves as either term E_1 or term E_2 depending on whether an action of E_1 or E_2 action is executed.

The related operational semantics of PA are shown in table 1.

Definition 2 A relation $B \subseteq \varsigma \times \varsigma$ is a weak bisimulation, if and only if, whenever $(E_1, E_2) \in B$ then for all a∈Act:

- whenever $E_1 \xrightarrow{a} E_1'$ then $E_2 \overset{\hat{a}}{\mapsto} E_2'$ and $(E_1', E_2') \in$ B;
- whenever $E_2 \xrightarrow{a} E_2'$ then $E_1 \overset{\hat{a}}{\mapsto} E_1'$ and $(E_1', E_2') \in$ B

The union of all weak bisimulations can be denoted as $\approx_B$. Here $\overset{\sigma}{\mapsto} \equiv \overset{\tau^m}{\Rightarrow} \overset{\sigma}{\Rightarrow} \overset{\tau^n}{\Rightarrow}$, and $\overset{\sigma}{\Rightarrow} \equiv \xrightarrow{a_1} \ldots \xrightarrow{a_n}$, if $\sigma = a_1 \ldots a_n$ For $\hat{a}$, if $a = \tau$, $\hat{a} = \varepsilon$ else $\hat{a} = a$

Definition 3 The state transition of **a** term E can be defined as: $E \equiv S_0 \xrightarrow{a_1} S_1 \ldots \xrightarrow{a_{n-1}} S_{n-1} \xrightarrow{a_n} S_n$ if there exists a executed trace $< a_1, a_2, .. a_n >$ for E, and the actions of this trace can be observed.

Deadlock in the process reflects that a process is in a blocked state where the process is not terminated successfully, and it can't continue to execute any action. The deadlock-free process can be formally defined as following:

Definition 4 A term E is said to be deadlock free, if and only if, for each state s of its underlying state transition graph, there exist an observable action a and a state s', such that $s \overset{a}{\Rightarrow} s'$

A simulation relation between two processes is formally defined as:

Definition 5 A relation $B \subseteq \varsigma \times \varsigma$ is a simulation, if and only if for $(E_1, E_2) \in B$, whenever $E_1 \overset{\sigma}{\mapsto} E_1'$, there exists E_2', $E_2 \overset{\sigma}{\mapsto} E_2'$ and $(E_1', E_2') \in B$.

3 System Model

In current component-based system, component model describes the provided and requested interfaces of a component, and its interactions with other ones through the interfaces, and etc. We present a formal component model with expansion of behavioral protocol as following:

1. $a.E \xrightarrow{a} E$
2. $\dfrac{E \xrightarrow{a} E'}{E/L \xrightarrow{a} E'/L}$ if $a \notin L$
3. $\dfrac{E \xrightarrow{a} E'}{E/L \xrightarrow{\tau} E'/L}$ if $a \in L$
4. $\dfrac{E_1 \xrightarrow{a} E_1'}{E_1 \|_S E_2 \xrightarrow{a} E_1' \|_S E_2}$ if $a \notin S$
5. $\dfrac{E_2 \xrightarrow{a} E_2'}{E_1 \|_S E_2 \xrightarrow{a} E_1 \|_S E_2'}$ if $a \notin S$
6. $\dfrac{E_1 \xrightarrow{a} E_1', E_2 \xrightarrow{a} E_2'}{E_1 \|_S E_2 \xrightarrow{a} E_1' \|_S E_2'}$ if $a \in S$
7. $\dfrac{E_1 \xrightarrow{a} E'}{E_1 + E_2 \xrightarrow{a} E'}$
8. $\dfrac{E_2 \xrightarrow{a} E'}{E_1 + E_2 \xrightarrow{a} E'}$
9. $\dfrac{E \xrightarrow{a} E'}{E[\varphi] \xrightarrow{\varphi(a)} E'[\varphi]}$
10. $\dfrac{E \xrightarrow{a} E'}{A \xrightarrow{a} E'}$ if $A \triangleq E$

Table 1 Operational semantics for PA

Definition 6 A component C with expansion of behavioral protocol is defined as such a tuple, $C = < I_C^P, I_C^R, A_C, L_C, P_C >$, where:

- I_C^P is a set of interfaces provided by C. For any $Ite_i \in I_C^P$, Ite_i provides a set of operations that are called by other components;
- I_C^R is a set of requested interfaces. For any $Ite_i \in I_C^R$, Ite_i contains a set of operations which are requested by other components;
- A_C is the set of executed actions, including three parts, requested, provided and internal actions, which are denoted as A_C^R, A_C^P and A_C^H respectively. These three parts are disjoint;
- L_C is a set of connections between C and other components. For any connection $l_i \in L_c$, $l_i =< RIte, PIte, Ins, PL >$ which denotes that C is connected with some other component. In the tuple, $RIte \in I_C^R$ is one of C's requested interfaces, $PIte$ is the corresponding interface provided by the external component C_i and $RIte$ and $PIte$ are matched in syntax. Hence, C is able to call C_i through $PIte$. The item Ins is an instance of C_i and PL is the location of C_i.
- P_C denotes the behavioural protocol of C, which is formal defined by PA. P_C is defined in such a tuple (S_C, Γ_C), where S_C a finite set of states. We use s_{Init} and s_{Fina} denote the initial and terminative state, respectively. $\Gamma_C \subseteq S_C \times A_C \times S_C$ is a finite set of transitions between states, such that $\Gamma_C = \{(s_i, a, s_j) \mid s_i, s_j \in S_C, a \in A_C \wedge s_i \xrightarrow{a} s_j\}$.

This paper supposes that P_C is defined as preserving determinacy, which means that if $P_C \overset{\sigma}{\Rightarrow} P_C'$ and $P_C \overset{\sigma}{\Rightarrow} P_C''$ then it holds $P_C' \approx_B P_C''$. P_C also holds correctly-terminated, which is for any state $s' \in S_C$, whenever $s' \neq s_{Fina}$, there exist a $\sigma \in A_C^*$ and $s' \overset{\sigma}{\Rightarrow} s_{Fina}$. We also use $trace_{co}(P_C)$ denote $trace_{co}(P_C) = \{\sigma \mid \sigma \in A_C^* \text{ and } s_{Init} \overset{\sigma}{\Rightarrow} s_{Fina}\}$.

For two components $C_i =< I_{C_i}^P, I_{C_i}^R, A_{Ci}, L_{Ci}, P_{C_i} >$ and $C_j =< I_{C_j}^P, I_{C_j}^R, A_{Cj}, L_{Cj}, P_{C_j} >$ connected with each other through an interface Ite provided by C_j, we will analyze their interaction behavior, $P_{C_i} \|_{Ite} P_{C_j}$. If C_i and C_j are connected through more than one interfaces, we use $e(C_i, C_j)$ denote the set of these interfaces, and the actions in these interfaces are denoted as $a(C_i, C_j) = \bigcup_{1 \le k \le |e(C_i,C_j)|} Ite_k$ $(Ite_k \in e(C_i, C_j))$. Clearly, if two components are disconnected, it holds that $e(C_i, C_j) = \phi$. During the interactions, C_i and C_j will asynchronously execute the actions belonging to $a(C_i, C_j)$. Based on this, we present the notion of behavioral compatibility as following:

Definition 7 For two components $C_i = < I_{C_i}^P, I_{C_i}^R, A_{Ci}, L_{Ci}, P_{C_i} >$ and $C_j = < I_{C_j}^P, I_{C_j}^R, A_{Cj}, L_{Cj}, P_{C_j} >$ assembled together through an interface Ite, C_i and C_j is behavioral compatibility on Ite if and only if, for the initial states of $P_{C_i} \|_{Ite} P_{C_j}$, (s_{Init}^i, s_{Init}^j), there exists a $\sigma \in (A_{C_i} \cup A_{C_j})^* \wedge \sigma \uparrow Ite \neq < >$, and it holds that $(s_{Init}^i, s_{Init}^j) \overset{\sigma}{\Rightarrow} (s_{Fina}^i, s_{Fina}^j)$.

A software system containing n components $\{C_1, C_2, \ldots, C_n\}$ $(C_i =< I_{C_i}^P, I_{C_i}^R, A_{C_i}, L_{C_i}, P_{C_i} >)$ is formally defined is as follows:

Definition 8 A software system, MCS, containing n component, is denoted as $MCS =< V_M, E_M, I_M, T_M, P_M >$, where:

- $V_M = \{C_1, C_2, \ldots, C_n\}$ is a set of software components assembled in the system, and $C_i = < I_{C_i}^P, I_{C_i}^R, A_{Ci}, L_{Ci}, P_{C_i} >$;
- $E_M = \bigcup_{1 \le i,j \le n, i \neq j} e(C_i, C_j)$ is the set of all interfaces through which the components interact with each other;
- I_M is the set of interfaces through which MCS interacts with its external environment. For $I_M = I_M^P \cup I_M^R$, $I_M^P = \left(\bigcup_{1 \le i \le n} I_{C_i}^P\right) \Big/ E_M$ is the set of interfaced provided by MCS, and $I_M^R = \left(\bigcup_{1 \le i \le n} I_{C_i}^R\right) \Big/ E_M$ is the set of interfaces requested by it;
- T_M is a set of all connections in MCS, $T_M = \bigcup_{1 \le i \le n} L_{C_i}$;
- P_M is the internal behavior of MCS and it is defined as as $P_M = P_{C_1} \|_{a(C_1,C_2)} P_{C_2} \|_{a(C_1,C_3) \cup a(C_2,C_3)} P_{C_3} \| \ldots \|_{a(C_1,C_n) \cup \ldots \cup a(C_{n-1},C_n)} P_{C_n}$.

The state s_M of P_M is denoted as $s_M = (s_1, \ldots, s_n)$ where $s_i \in S_{C_i}$, $1 \le i \le n$. Then, the state transitions of P_M change as following:

Definition 9 Suppose s and s' are two different states of P_M, $s = (s_1, \ldots, s_n)$ and $s' = (s_1', \ldots, s_n')$. When one of the two following conditions are satisfied, P_M transits from state s to state s' by executing an action a, which is $s \xrightarrow{a} s'$.

- there are an action $a \in A_{C_i}^H$ and such a state transition $(s_i, a, s_i') \in \Gamma_{C_i}$, and for the states of the other component $C_j (1 \le j \le n, j \ne i)$, it holds $s_j = s_j'$;
- there are an action $a \in Ite$, $Ite \in e(C_i, C_j)$ $(1 \le i, j \le n, i \ne j)$, and two state transitions $(s_i, a, s_i') \in \Gamma_{C_i}$ and $(s_j, a, s_j') \in \Gamma_{C_j}$, and for the states of the other components C_k $(1 \le k \le n, k \ne i, j)$, it holds that $s_k = s_k'$.

The deadlock-free behavior of *MCS*, P_M, means that executing any synchronous actions in the interfaces makes none of the components into a deadlock state. A system with a deadlock-free behavior can be formally defined as following:

Definition 10 For a *MCS* containing n components, $\{C_1, C_2, \ldots, C_n\}$ $(C_i = < I_{C_i}^P, I_{C_i}^R, A_{C_i}, L_{C_i}, P_{C_i} >)$, its behavior P_M is deadlock-free, if and only if, for P_M's Initial state $(s_{Init}^1, s_{Init}^2, \ldots, s_{Init}^n)$, there exists a trace $\sigma \in (\bigcup_{1 \le i \le n} A_{C_i})^*$ with $\sigma \uparrow (\bigcup_{1 \le i, j \le n, i \ne j} a(C_i, C_j)) \ne <>$, and it holds that $(s_{Init}^1, s_{Init}^2, \ldots, s_{Init}^n) \stackrel{\sigma}{\Rightarrow} (s_{Fina}^1, s_{Fina}^2, \ldots, s_{Fina}^n)$.

For a software components system *MCS* holding a deadlock-free behavior, if one of its components C_j is replaced by a new one C_j', and C_j' still preserves behavioral compatibility with other components in the system, we say that C_j can be replaced by C_j'. Suppose a component $C_i = < I_{C_i}^P, I_{C_i}^R, A_{C_i}, L_{C_i}, P_{C_i} >$ will be replaced by a new one $C_i' = < I_{C_i'}^P, I_{C_i'}^R, A_{C_i'}, L_{C_i'}, P_{C_i'} >$. To meet the requirement of replacement, a necessary condition must be holding, which is $I_{C_i'}^R \subseteq I_{C_i}^R \wedge A_{C_i'}^R \subseteq A_{C_i}^R$. It means that the new component can't require more things from the environment than the old one.

4 Verification of Behavioural Substitutability

4.1 Behavioral Compatibility Between Two Components

Suppose component C_i interacts with C_j through an interface *Ite* provided by C_j. To preserve behavioral compatibility, C_i only concerns the behavior of C_j shown on *Ite*. The same to C_j, it only concerns whether the corresponding request behavior of C_i showing on the interface meets its requirements. So, if C_j can support all the requests from C_i on this interface, and simultaneously, the behavior of C_i meets the conditions needed by C_j, then they will interact with each other compatibly.

Theorem 1 Suppose two components $C_i = < I_{C_i}^P, I_{C_i}^R, A_{Ci}, L_{Ci}, P_{C_i} >$ and $C_j = < I_{C_j}^P, I_{C_j}^R, A_{Cj}, L_{Cj}, P_{C_j} >$ interact with each other through an interface provided by C_j, *Ite*. If the two following conditions hold, the behavior of the interactions between C_i and C_j on *Ite* is compatible:

- $(P_{C_i} \,||_{Ite}\, P_{C_j})/D1 \approx_B P_{C_i}/D2$ here $D1 = A_{C_i} \cup A_{C_j} - Ite$, $D2 = A_{C_i} - Ite$
- $trace_{co}(P_{C_i}) \uparrow Ite \subseteq trace_{co}(P_{C_j}) \uparrow Ite$;

The proof of theorem 1 is in [15], which gives a condition of a partial order behavioral compatibility between two components interaction. In this scene, only one component has requests for the other. While in some scenes, both of the two components have requests for the other. Suppose two components C_i and C_j interact with each other through the interfaces Ite_1 and Ite_2 provided by C_i and C_j, respectively. If each component can meet the requirements of the other one on the provided interface, and follow the conditions needed by the other one on the requested interface, they will interact with each other compatibly.

Theorem 2 Suppose two components $C_i = < I_{C_i}^P, I_{C_i}^R, A_{C_i}, L_{C_i}, P_{C_i} >$ and $C_j = < I_{C_j}^P, I_{C_j}^R, A_{C_j}, L_{C_j}, P_{C_j} >$ interact with each other through two interfaces Ite_1 and Ite_2, $Ite_1 \in I_{C_i}^P \cap I_{C_j}^R \wedge Ite_2 \in I_{C_i}^R \cap I_{C_j}^P$, which are provided by C_i and C_j, respectively. If the following two conditions hold simultaneously, the behavior of the interactions between C_i and C_j on the interfaces Ite_1 and Ite_2 is compatible:

- ($P_{C_i} \|_{Ite_1 \cup Ite_2} P_{C_j}$)/$D1 \approx_B P_{C_i}$ /$D2$, and $trace_{co}(P_{C_i}) \uparrow Ite_2 \subseteq trace_{co}(P_{C_j}) \uparrow Ite_2$, where $D1 = (A_{C_i} \square A_{C_j}) \square Ite_2 \square D2 = A_{C_i} \square Ite_2$;
- ($P_{C_i} \|_{Ite_1 \cup Ite_2} P_{C_j}$)/$D3 \approx_B P_{C_j}$ /$D4$, and $trace_{co}(P_{C_j}) \uparrow Ite_1 \subseteq trace_{co}(P_{C_i}) \uparrow Ite_1$, where $D3 \square (A_{C_i} \square A_{C_j}) \square Ite_1 \square D4 = A_{C_j} \square Ite_1$.

The proof of theorem 2 is also in [15].

4.2 Behavioral Substitutability in a Two-Component Assembly

Suppose two components C_i and C_j interact with each other through an interface *Ite* provided by C. Their behavioral compatibility follows the conditions presented in theorem 1. Now, *C* will be updated by a new component *C'* that has a new provided interface *Ite*' corresponding to *Ite*. To preserve behavioral compatibility in the updated system, three conditions must be satisfied simultaneously: 1) All the operations provided in *Ite* must also be provided in *Ite*', which means *Ite* $\subseteq$ *Ite*' ; 2) To meet the external requirements from other components, all the behavior of *C* shown on *Ite* will be also supported by *C'*; 3) As *Ite*' may contain some new operations that don't appear in *Ite*, the execution of these new operations will not affect the execution of the old operations. We present this rule as following:

Theorem 3 Suppose two components $C_i \square < I^P_{C_i}, I^R_{C_i}, A_{C_i}, L_{C_i}, P_{C_i} >$ and $C_j \square < I^P_{C_j}, I^R_{C_j}, A_{C_j}, L_{C_j}, P_{C_j} >$ are assembled through an interface *Ite* provided by C_j. Their behavioral compatibility follow the conditions presented in theorem 1. Now, C_j will be updated by a new one $C_j' = < I^P_{C_j'}, I^R_{C_j'}, A_{C_j'}, L_{C_j'}, P_{C_j'} >$ with a provided interface *Ite*' corresponding to *Ite*, and *Ite* $\subseteq$ *Ite*'. If it holds that ($P_{C_j'} \setminus (Ite' - Ite)$)/($A_{C_j'} - Ite'$) $\approx_B P_{C_j}$ /($A_{C_j} - Ite$), C_j can be behaviorally substituted by C_j'.

The proof of theorem 3 is in [16]. It shows the fact that *Ite*' provided by the new component may contain a set of new operations, *Ite*' $-$*Ite*. And execution of these new operations will not influence the executions of other operations.

Often, the new component doesn't include additional operations in *Ite*', which means *Ite*' $\square$ *Ite* $= \phi$. It may just extend the provided behavior on the interface. However, theorem 3 can't verify this case. Here, we first introduce a notion of dual component, and based on it we present another rule to verify the behavioral compatibility under this case.

Definition 11 Component $\overline{C}$ is a dual component of *C,* if the following conditions hold (1) $A^R_{\overline{C}} = A^P_C \square A^P_{\overline{C}} = A^R_C \square A^H_{\overline{C}} = A^H_C$; (2) $I^R_{\overline{C}} = I^P_C \square I^P_{\overline{C}} = I^R_C$; 3) $P_{\overline{C}} \equiv P_C$.

From the definition, we can see that the behavioral protocol of $\overline{C}$ is the same as the one of *C*, except that all requested operations become provided operations, and all requested operations become provided operations in the dual component.

Lemma 4 Suppose two components $C_i \square < I^P_{C_i}, I^R_{C_i}, A_{C_i}, L_{C_i}, P_{C_i} >$ and $C_j \square < I^P_{C_j}, I^R_{C_j}, A_{C_j}, L_{C_j}, P_{C_j} >$ interact with each other through an interface *Ite* provided by C_j . Their behavioral compatibility satisfies the conditions presented in theorem 1. Now, suppose $\overline{C}_j$ is the dual component of component C_j , $P_{\overline{C}_j}$ /D can simulate P_{C_i} /$D2$, where $D = A_{\overline{C}_j} - Ite$.

The proof of lemma 4 is presented in [16]. From the lemma 4, if the behavior of the interactions between C_i and C_j are compatible on *Ite*, and the request behavior of C_i on *Ite* is just a subset of the one of C_j . Hence, we can analyze the requested behavior of C_i on *Ite*, by the behavior of its dual component $\overline{C}_j$. Obviously, for the new component C_j', if it meets the behavioral requirements requested by $\overline{C}_j$ on *Ite* $\square$ it can also satisfy all the possible behavioral requirements requested by C_i on *Ite.* The rule is presented as following

Theorem 5 Suppose two components $C_i \square < I^P_{C_i}, I^R_{C_i}, A_{C_i}, L_{C_i}, P_{C_i} >$ and $C_j \square < I^P_{C_j}, I^R_{C_j}, A_{C_j}, L_{C_j}, P_{C_j} >$ interact with each other through an interface *Ite* provided by C_j . Their behavioral compatibility follow the conditions presented in theorem 1. Now, a new component $C_j' \square < I^P_{C_j'}, I^R_{C_j'}, A_{C_j'}, L_{C_j'}, P_{C_j'} >$ will be used to replace C_j , and C_j' has a provided interface *Ite*' corresponding to *Ite*. Let $\overline{C}_j$ be the dual component of C_j . If $P_{\overline{C}_j}$ and $P_{C_j'}$ satisfy the following conditions: 1)($P_{\overline{C}_j} \|_{Ite'} P_{C_j'}$)/$D \approx_B P_{\overline{C}_j}$ /D', where $D = (A_{\overline{C}_i} \cup A_{C_j'}) \square Ite'$, $D' = A_{\overline{C}_j} \square Ite'$; and 2) $trace_{co}(P_{\overline{C}_j}) \uparrow Ite' \subseteq trace_{co}(P_{C_j'}) \uparrow Ite'$, then C_j can be behaviorally substituted by C_j'.

The proof of theorem 5 is presented in [16]. It can be used to verify the behavioral substitutability when the new component may extend both its operations and behavior on the interface simultaneously. However, in an assembly containing two components C_i and C_j, where both components have requests for the other, theorem 3 and 5 can't verify the behaviorally substitutability in the case. In this case, if a new component C_j' is able to replace C_j, its behavior of requirements from other components can't be expanded more than the one of C_j, and its behavior of provision to other components can't be weaken less than the one of C_j simultaneously. In this way, C_i can meet the requirements of C_j' and C_j' can satisfy the requirements of C_i at the same time. A rule to verify the substitutability under this scenario is given as following:

Theorem 6 Suppose two components $C_i = < I^P_{C_i}, I^R_{C_i}, A_{C_i}, L_{C_i}, P_{C_i} >$ and $C_j = < I^P_{C_j}, I^R_{C_j}, A_{C_j}, L_{C_j}, P_{C_j} >$ interact with each other through two interfaces Ite_1 and Ite_2, $Ite_1 \in I^P_{C_i} \cap I^R_{C_j}$ and $Ite_2 \in I^R_{C_i} \cap I^P_{C_j}$. Their behavioral compatibility satisfies the conditions presented in theorem 2. Now, a new component $C_j' = < I^P_{C_j'}, I^R_{C_j'}, A_{C_j'}, L_{C_j'}, P_{C_j'} >$ is use to replace C_j. Its new provided interface is Ite_2' corresponding to Ite_2, with $Ite_2 \subseteq Ite_2'$, and its requested interface is still Ite_1. Let $\overline{C}_j$ be the dual component of component C_j, if $P_{\overline{C}_j}$ and $P_{C_j'}$ satisfy the following conditions:

- $(P_{C_j'} \|_{Ite_1 \cup Ite_2'} P_{\overline{C}_j}) / (A_{C_j'} \cup A_{\overline{C}_j} \setminus Ite_2') \approx_B P_{\overline{C}_j} / (A_{\overline{C}_j} \setminus Ite_2')$, and $trace_{co}(P_{\overline{C}_j}) \uparrow Ite_2' \subseteq trace_{co}(P_{C_j'}) \uparrow Ite_2'$;
- $(P_{C_j'} \|_{Ite_1 \cup Ite_2'} P_{\overline{C}_j}) / (A_{C_j'} \cup A_{\overline{C}_j} \setminus Ite_1) \approx_B P_{C_j'} / (A_{C_j'} \setminus Ite_1)$, and $trace_{co}(P_{C_j'}) \uparrow Ite_1 \subseteq trace_{co}(P_{\overline{C}_j}) \uparrow Ite_1$.

then C_j can be behaviorally substituted by C_j'

The proof of theorem 6 is presented in [16].

4.3 Behavioral Substitutability in the System Containing Multiple Components

In the current component-based software system, a component may interact with multiple components through different interfaces. In this scenario, we also study the difference of the interaction behavior between the new component and the old one replaced, and then present our verification rules. Let $P_M \equiv P_{C_1} \|_{a(C_1,C_2)} P_{C_2} \|_{a(C_1,C_3) \cup a(C_2,C_3)} P_{C_3} \|... \|_{a(C_1,C_n) \cup ... \cup a(C_{n-1},C_n)} P_{C_n}$ denote the behavior of the system. Clearly, P_M can also be defined in such a form, $P_M \equiv P_{C_i} \|_{S_i} P^i$, where $S_i = \bigcup_{1 \le k \le n, k \ne i} a(C_k, C_i)$ and $P^i \equiv P_{C_1} \|_{a(C_1,C_2)} P_{C_2} \|_{a(C_1,C_3) \cup a(C_2,C_3)} P_{C_3} \|.. \|_{a(C_1,C_{i-1}) \cup ... \cup a(C_{i-2},C_{i-1})} P_{C_{i-1}} \|_{a(C_1,C_{i+1}) \cup ... \cup a(C_{i-1},C_{i+1})} P_{C_{i+1}} \|... \|_{a(C_1,C_n) \cup ... \cup a(C_{n-1},C_n)} \|_{a(C_1,C_n) \cup ... \cup a(C_{n-1},C_n)} P_{C_n}$. Let C_i be such a component that it provides several interfaces to other components for use and has no requirements for other ones. Now, C_i will be updated by a new one C_i'. Obviously, if C_i' can support all the provided behavior supported by C_i, and still has no requirements for other ones simultaneously, C_i can be replaced by C_i' successfully.

Theorem 7 Suppose a *MCS* contain a set of n components, $\{C_1, C_2, ..., C_n\}$ ($C_i \setminus < I^P_{C_i}, I^R_{C_i}, A_{Ci}, L_{Ci}, P_{C_i} >$) and its behavior P_M be $P_M = P_{C_1} \|_{a(C_1,C_2)} P_{C_2} \|_{a(C_1,C_3) \cup a(C_2,C_3)} P_{C_3} \|... \|_{a(C_1,C_n) \cup ... \cup a(C_{n-1},C_n)} P_{C_n}$. There is a component C_i in *MCS* with $I^R_{C_i} = \phi$, and its behavior P_{C_i} satisfying the following conditions: 1)$(P_{C_i} \|_{S_i} P^i) / D \approx_B P^i / D'$, where $D = \bigcup_{1 \le k \le n} A_{C_k} \setminus S_i$ and $D' = \bigcup_{1 \le k \le n, k \ne i} A_{C_k} \setminus S_i$; and 2) $trace_{co}(P^i) \uparrow S_i \subseteq trace_{co}(P_{C_i}) \uparrow S_i$. Now, a new component $C_i' \setminus < I^P_{C_i'}, I^R_{C_i'}, A_{C_i'}, L_{C_i'}, P_{C_i'} >$ is used to replace C_i, and it holds that $I^P_{C_i} \subseteq I^P_{C_i'}$ and $A^P_{C_i} \subseteq A^P_{C_i'}$. Let $S_i' = \bigcup_{1 \le k \le n, k \ne i} a(C_k, C_i')$, and $\overline{C}_j$ be the dual component of C_j. If $P_{\overline{C}_j}$ and $P_{C_j'}$ satisfy the following conditions:

- $(P_{\overline{C}_i} \|_{S_i'} P_{C_i'}) / D1 \approx_B P_{\overline{C}_i} / D2$, where $D1 = A_{\overline{C}_i} \cup A_{C_i'} \setminus S_i'$ and $D2 = A_{\overline{C}_i} \setminus S_i'$;
- $trace_{co}(P_{\overline{C}_i}) \uparrow S_i' \subseteq trace_{co}(P_{C_i'}) \uparrow S_i'$;

then C_i can be behaviorally substituted by C_i' in *MCS*.

The proof of theorem 7 is presented in [16]. In another scenario, a component may have both the provided and requested behavior to interact with other components. We take the behavioral compatibility of these two aspects into account. Obviously, compared to the behavior of the old component, if a new component can take the place of

the old one, its behavior of requirements for other components can't be expanded, and its behavior of provision to other components can't be weaken simultaneously. Based on this, we present the following rule:

Theorem 8 Suppose a *MCS* contain a set of n components, $\{C_1, C_2, ..., C_n\}$(C_i □ $< I^P_{C_i}, I^R_{C_i}, A_{Ci}, L_{Ci}, P_{C_i} >$), and its behavior P_M be $P_M = P_{C_1} \|_{a(C_1,C_2)} P_{C_2} \|_{a(C_1,C_3)\cup a(C_2,C_3)} P_{C_3} \| ... \|_{a(C_1,C_n)\cup...\cup a(C_{n-1},C_n)} P_{C_n}$. There is a component C_i in *MCS* with $I^R_{C_i} \neq \phi$ □ $I^P_{C_i} \neq \phi$. Let $S_i = \bigcup_{1\le k\le n, k\neq i} a(C_k, C_i)$, $S^R_i = S_i \cap \bigcup_{1\le k\le |I^R_{C_i}|} Ite_k$ ($Ite_k \in I^R_{C_i}$), and $S^P_i = S_i \cap \bigcup_{1\le k\le |I^P_{C_i}|} Ite_k$ ($Ite_k \in I^P_{C_i}$). The behavior of C_i, P_{C_i} meets the following conditions: 1) ($P_{C_i} \|_{S_i} P^i$)/($\bigcup_{1\le k\le n} A_{C_k}$ □ S^P_i)≈B P^i /($\bigcup_{1\le k\le n, k\neq i} A_{C_k}$ □ S^P_i), and $trace_{co}$(P^i)↑ $S^P_i \subseteq trace_{co}$(P_{C_i})↑ S^P_i ; 2) ($P_{C_i} \|_{S_i} P^i$)/($\bigcup_{1\le k\le n} A_{C_k}$ □ S^R_i)≈B P_{C_i} /(A_{C_i} □ S^R_i), and $trace_{co}$(P_{C_i})↑ $S^R_i \subseteq trace_{co}$(P^i)↑ S^R_i . Now, a new component C_i' □ $< I^P_{C_i'}, I^R_{C_i'}, A_{C_i'}, L_{C_i'}, P_{C_i'} >$ is used to replace C_i, and it holds that $I^P_{C_i} \subseteq I^P_{C_i'} \wedge A^P_{C_i} \subseteq A^P_{C_i'}$ and $I^R_{C_i'} \subseteq I^R_{C_i} \wedge A^R_{C_i'} \subseteq A^R_{C_i}$. Let S_i' □ $\bigcup_{1\le k\le n, k\neq i} a(C_k, C_i')$, $S^P_{i'} = S_i' \cap \bigcup_{1\le k\le |I^P_{C_i'}|} Ite_k$ ($Ite_k \in I^P_{C_i'}$), and $\overline{C_i}$ be the dual component of C_i. If $P_{\overline{C_i}}$ and $P_{C_i'}$ satisfy the following conditions:

- ($P_{C_i'} \|_{S_i'} P_{\overline{C_i}}$)/($A_{C_i'} \cup A_{\overline{C_i}}$ □ S^R_i)≈B $P_{C_i'}$ /($A_{C_i'}$ □ S^R_i), and $trace_{co}$($P_{C_i'}$)↑ $S^R_i \subseteq trace_{co}$($P_{\overline{C_i}}$)↑ S^R_i ;
- ($P_{C_i'} \|_{S_i'} P_{\overline{C_i}}$)/($A_{C_i'} \cup A_{\overline{C_i}}$ □ $S^P_{i'}$)≈B $P_{\overline{C_i}}$ /($A_{\overline{C_i}}$ □ $S^P_{i'}$), and $trace_{co}$($P_{\overline{C_i}}$)↑ $S^P_{i'} \subseteq trace_{co}$($P_{C_i'}$)↑ $S^P_{i'}$;

then C_i can be behaviorally substituted by C_i' in *MCS*.

The proof of theorem 8 is presented in [16].

5 An e-commerce Example

In this paper, we express the characteristics of the rules through a specific example of e-commerce. In this example, the persons buy books on an e-commerce system, and three components are included in this system, *BookShop* (C_{BS}), *BookBroker* (C_{BB}) and *Bank* (C_{BA}). In the system□ the component *BookShop* registers at *BookBroker* first. When a user wants to buy books, it will call the interface operation *getABook* provided by component *BookBroker*. *BookBroker* inquires *BookShop* whether there is the book in stock by calling the interface *inStock* provided by *BookShop*. If *BookShop* has the book, *BookBroker* will order the book by calling the operation *Order*, allow *BookShop* deliver books to *User* by calling the operation *deliver*, and deposit money in *BookShop's* bank account by calling the operation *deposit* provided by *Bank*. Their assembly structure, interfaces described in IDLs and definitions are given in Figure 1~3. Obviously, the behavioral compatibility between C_{BS} and C_{BB} on interfaces Ite_{Book_Shop} and Ite_{Broker_Shop} satisfies the conditions in theorem 2, and the behavioral compatibility between C_{BB} and C_{BA} on interfaces $Ite_{BankAccount}$ satisfies the conditions presented in theorem 1.

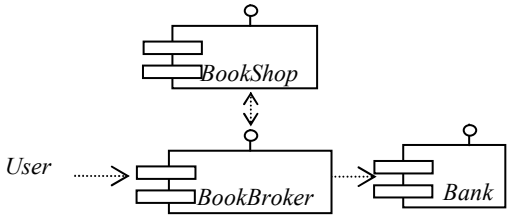

Fig.1 An e-commerce instance

Now, the system is upgraded to provide more functions. Two new components *Bank*' and *Bookshop*' are used to replace the old ones, respectively. The IDL interfaces, and formal definitions of these new components are shown in Figure 4 and 5. Component *Bank*' add a new operation *Query* in its provided interface, which will be used to query the client's deposit. Component *Bookshop*' add a new function to cancel the orders of books in its interface, *cancelOrder*. Their behavioral is also adapted. Now we will verify whether the new components *Bank*' and *Bookshop*' can replace the old ones successfully. In the upgraded system, component *Bank*' only interacts with *BookBroker* through the interface, and it has no requests for other components in the environment. Theorem 3 and 5 can verify the behavioral compatibility under this scenario. Clearly, it holds that ($P_{C_{BA'}} \backslash D1$)/$D2$≈B $P_{C_{BA}}$ /$D3$, where $D1=Ite_{BankAccount}'$ □ $Ite_{BankAccount}$ $=\{query, query\_r\}$, $D2= A_{C_{BA'}} - Ite_{BankAccount}' \neq \phi$ and $D3= A_{C_{BA}} - Ite_{BankAccount} = \phi$. It meets the conditions presented in theorem 3. On the other hand, component *Bookshop*' only interacts with *BookBroker* through two interfaces. We will use the theorem 6 to verify the behavioral compatibility under this scenario.

An interface provided by Bank : *BankAccount*

```
interface BankAccount{
  void  login( in string accountNO );
  float getBalance();
  string deposit( in float amount);
  string withdraw( in float amount);
  void logout();
};
```

An interface provided by BookShop: *Book_Shop*

```
interface Book_Shop{
  struct  BookRef { string ISBN, float price; }
  boolean  inStock(in string title, in string author);
  void    order( in BookRef  b, out account a, out string purchaseID);
  date   deliver( in string purchaseID, in string rcpt, in string addr);
  };
```

An interface provided by BookBroker: *Broker_User*

```
interface Broker_User{
boolean getABook( in string author, in string title,
                  in float maxprice, in string addr,
                  out date when);
};
```

An interface provided by BookBroker: *Broker_Shop*

```
interface Broker_Shop{
  void register ( in Bookshop b);
  void unregister ( in Bookshop b);
};
```

Fig.2 IDL interfaces provided by components *Bank*、*BookShop* and *BookBroker*

BookShop、C_{BS} 、$< I^P_{C_{BS}}, I^R_{C_{BS}}, A_{C_{BS}}, L_{C_{BS}}, P_{C_{BS}} >$

$I^P_{C_{BS}}$: { Ite_{Book_Shop} }; Ite_{Book_Shop} = {*inStock, inStock_r, order, deliver, deliver_r*}

$I^R_{C_{BS}}$: { Ite_{Broker_Shop} }; Ite_{Broker_Shop}= { *register, unregister* }

$A_{C_{BS}}$: ($A^R_{C_{BS}}, A^P_{C_{BS}}, A^H_{C_{BS}}$);

$A^R_{C_{BS}}$: { *register, unregister*}; $A^H_{C_{BS}} : \phi$;

$A^P_{C_{BS}}$: { *inStock, inStock_r, order, deliver, deliver_r* };

$L_{C_{BS}}$: {< Ite_{Broker_Shop}, Ite_{Broker_Shop}, *BookBroker_Inst*, *BookBroker _Host* >};

$P_{C_{BS}}$: { $P_{C_{BS}} \triangleq P[Shop]_{Init}$

$P[Shop]_{Init} \triangleq register.P[Shop]_1$

$P[Shop]_1 \triangleq inStock.inStock\_r.P[Shop]_2$

$P[Shop]_2 \triangleq order.deliver.deliver\_r.P[Shop]_1 + unregister.P[Shop]_{Fina}$

$P[Shop]_{Fina} \triangleq \underline{0}$ }、

BookBroker: C_{BB} 、$< I^P_{C_{BB}}, I^R_{C_{BB}}, A_{C_{BB}}, L_{C_{BB}}, P_{C_{BB}} >$

I^P_{BB} :{ Ite_{Broker_User}, Ite_{Broker_Shop} }; Ite_{Broker_User}={ *getABook, getABook_r*}; Ite_{Broker_Shop}={ *register, unregister*};

$I^R_{C_{BB}}$: { Ite_{Book_Shop}, $Ite_{BankAccount}$ }; Ite_{Book_Shop} = {*inStock, inStock_r, order, deliver, deliver_r*};

$Ite_{BankAccount}$ ={*login, getBalance, getBalance_r, deposit, deposit_r, withdraw, withdraw_r,logout*}

$A_{C_{BB}}$: ($A^R_{C_{BB}}, A^P_{C_{BB}}, A^H_{C_{BB}}$);

$A^R_{C_{BB}}$: { *inStock, inStock_r, order, login, deposit,deposit_r, logout, deliver, deliver_r* }; $A^H_{C_{BB}} : \phi$;

$A^P_{C_{BB}}$: { *getABook, getABook_r, register, unregister*};

$L_{C_{BB}}$: {< Ite_{Book_Shop}, Ite_{Book_Shop}, *BookShop_Inst, BookShop_Host* >, < $Ite_{BankAccount}$, $Ite_{BankAccount}$, *Bank_Inst, Bank_Host* >};

$P_{C_{BB}}$: { $P_{C_{BB}} \triangleq P[Broker]_{Init}$

$P[Broker]_{Init} \triangleq register.P[Broker]_1$

$P[Broker]_1 \triangleq getABook.P[Broker]_2$

$P[Broker]_2 \triangleq inStock.inStock\_r.getABook\_r.P[Broker]_3$

$P[Broker]_3 \triangleq order.P[Broker]_4 + unregister.P[Broker]_{Fina}$

$P[Broker]_4 \triangleq login.deposit.deposit\_r.logout.deliver.deliver\_r.P[Broker]_1$

$P[Broker]_{Fina} \triangleq \underline{0}$ }、

Bank : C_{BA} 、$< I^P_{C_{BA}}, I^R_{C_{BA}}, A_{C_{BA}}, L_{C_{BA}}, P_{C_{BA}} >$

$I^P_{C_{BA}}$: { $Ite_{BankAccount}$}; $Ite_{BankAccount}$={*login, getBalance, getBalance_r, deposit, deposit_r, withdraw, withdraw_r, logout*};

$I^R_{C_{BA}} : \phi$;

$A_{C_{BA}}$: ($A^R_{C_{BA}}, A^P_{C_{BA}}, A^H_{C_{BA}}$);

$A^R_{C_{BA}}$: ϕ ; $A^H_{C_{BA}}$: ϕ

$A^P_{C_{BA}}$: { *login, getBalance,getBalance_r, deposit, deposit_r, withdraw, withdraw_r, logout* };

$L_{C_{BA}} : \phi$;

$P_{C_{BA}} : \{ P_{C_{BA}} \triangleq P[Bank]_{Init}$

$P[Bank]_{Init} \triangleq login.P[Bank]_1$

$P[Bank]_1 \triangleq getBalance.\ getBalance\_r.P[Bank]_1 + deposit.\ deposit\_r.P[Bank]_1 + withdraw.$
$withdraw\_r.P[Bank]_1 + logout.P[Bank]_{Fina.}$

$P[Bank]_{Fina.} \triangleq \underline{0} \}$

Fig.3 Specifications of components *Bank*, *BookShop* and *BookBroker*

Clear, we can see that the following two conditions hold: 1) $(P_{\overline{C}_{BS}} \|_{Ite1 \square Ite2'} P_{C_{BS}'})/D3 \approx_B P_{\overline{C}_{BS}}/D4$, where $Ite1=Ite_{Broker\_Shop}$, $Ite2'= Ite_{Book\_Shop}'$, $D3= A_{C_{BS}'} \cup A_{\overline{C}_{BS}} \square Ite_{Book\_Shop}'=\{register, unregister\}$, $D4= A_{\overline{C}_{BS}} \square Ite_{Book\_Shop}'=\{register, unregister\}$, and $trace_{co}(P_{\overline{C}_{BS}}) \uparrow Ite2' \subseteq trace_{co}(P_{C_{BS}'}) \uparrow Ite2'$; 2) $(P_{\overline{C}_{BS}} \|_{Ite1 \square Ite2'} P_{C_{BS}'})/D5 \approx_B P_{C_{BS}'}/D6$, where $D5= A_{C_{BS}'} \cup A_{\overline{C}_{BS}} \square Ite_{Broker\_Shop}=\{ inStock, inStock\_r, order, deliver, cancelOrder, cancelOrder\_r, deliver\_r\}$, $D6 \square A_{C_{BS}'} \square Ite_{Broker\_Shop}=\{ inStock, inStock\_r, order, deliver, cancelOrder, cancelOrder\_r, deliver\_r\}$, $trace_{co}(P_{C_{BS}'}) \uparrow Ite1 \subseteq trace_{co}(P_{\overline{C}_{BS}}) \uparrow Ite1$. Hence, the components can be replaced by the new ones successfully.

6 Related Works and Conclusion

At present, some research works focus on the behavioral compatibility in component-based systems. In [3], the authors describe and analyze the behavioral compatibility between two components by using π calculus, but they didn't taken into account the scenario where the new components may change its external behavior, nor did they take into account the scenario where a system may contain multiple components.

In [6], the authors concerned behavioral inheritance in component-based software system by using Petri Nets. They presented several rules for testing whether the components are suitable for the requirements of system by analyzing the correspondences between the component's external behavior and the descriptions of the system. They defined a type of inheritance called project inheritance. Through this inheritance, the system can ensure the component replaced meet the requirements of the system without impacting its external behavior. But in [6], the author didn't focus on the scenario where a new component has the external request behavior to the environment.

The similar works were also presented to ensure the feasibility of component replacement in [4], [5] and [7]. In these works, the author studied the behavioral subtype relationship between objects by using CSP, and presented three behavioral subtypes: weak subtype, safe subtype and optimization subtype. These three behavioral subtypes can be used to analyze object substitutability in the object-based system, but it has not yet consider scenes where software entities replaced can have external request behavior.

Current component technology and tools ensure the component substitution mainly through the IDL interface file, and they don't pay much attention to the property of behavior. From the middle of the 1990s to now, researchers have focused on the behavioral substitutability in component-based software system. This paper, based on the existing work, analyzes the behavior of the component in both their external requested and provided interfaces, and present a set of rules for the upgraded system still preserving deadlock-free characteristics. Our future work will focus on some other prospects of the behavioral compatibility in component-based software components, such as the description and verification of system performance, the Qos-based component assembly and substitutability, the type of connections among components, and etc.

Acknowledgement:

This paper is supported by National Natural Science Foundation of China granted by Nos. 60873022, the Science and Technology Foundation of Zhejiang province granted by Nos. 2008C23034, 2008C13082 and the open fund provided by State Key Laboratory for Novel Software Technology of Nanjing University, and the Natural Science Foundation of Zhejiang Province of China under Grant No.Y1080148.

References:

[1] Heineman GT, Councill WT. *Component-based software engineering*. Boston: Addison-Wesley, 2001.
[2] Szyperski C. *Component software: Beyond Object-Oriented programming.* Pearson Education Limited, 2003.
[3] Canal C, Pimentel E, Troya JM. Compatibility and inheritance in software architectures. *Science of Computer Programming*, 2001,41:105-138.
[4] Wehrheim, H. Relating State-based and Behaviour-oriented *Subtyping. Nordic Journal of Computing* 2002, 9(4),405-435
[5] Schrefl M. Stumptner M. Behavior-Consistent Specialization of Object Life Cycles. *ACM Transaction on Software Engineering and Methodology*, 11(1), 92-148,2002.
[6] Van Der Aalst W.M.P., Toorn R.A. Component-based software architectures: a framework based on inheritance of behavior. *Science of Computer Programming* 42:129-171, 2002.
[7] Wehrheim, H. Checking Behavioural Subtypes via Refinement. In FMOODS 2002: Formal methods for Open Object-Based Distributed Systems. Kluwer, 79-93.
[8] Bernardo M, Ciancarini P, Donatiello L. Architecting families of software systems with process algebras. *ACM Transaction on Software Engineering and Methodology*, 2002, 11(4): 386-426.
[9] Canal C, Pimentel E, Troya J. *Specification and refinement for dynamic software architectures. Software Architecture*. Netherlands. Kluwer Academic Publisher, 1999: 107-126.
[10] Allen R, Garlan D. A formal basis for architectural connection. *ACM Transaction on Software Engineering and Methodology*. 1997, 16(3):213-249.
[11] Bernardo M. *Theory and Application of Extended Markovian Process Algebra*.[Ph.D] Dottorato di Ricerca in Informatica, University di Bologna, Padova, Venezia. 1999.
[12] Mei H, Chen F, Feng YD, Yang J. ABC: An architecture based, component oriented approach to software development. *Journal of Software,* 2003,14(4):721-732(in Chinese with English abstract).
[13] Magee J, Kramer J. Dynamic structure in software architectures. *ACM SIGSOFT Software Engineering Notes*, 1996, 21(6):3-14.
[14] Hu HY, Yang M, Tao XP, Lu J. Research and implementation of late assembly technology in Cogent. *ACTA ELECTRONICA SINCA*, 2002,30(12): 1823- 1827(in Chinese with English abstract).
[15] Hu HY, Lu J, Ma XX, Tao XP. Research on Behavioral Compatibility of Components in Software Architecture Using Object-Oriented Paradigm *Journal of software*,2006,17(6):1276-1286.
[16] Haiyang Hu. *Study on the technology of softare component assembly*. Ph.D Dissertation. Nanjing University, 2006.