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

HAIYANG HU1,2 HUA HU1,3
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 practitioners 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:
\[ E::= \emptyset | a.E | E/L | E \oplus L | E[\sigma] | E_1 + E_2 | E_1 || E_2 | A \]

\[ B \]

- \( \emptyset \) 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 \( \tau \), whenever \( a \in L \).
- \( "E/\| L" \) behaves as term \( E \) except that each executed action \( a \) is forbidden, whenever \( a \in L \), and \( E/\| L \equiv E || \emptyset \).
- \( "E[\sigma]" \) behaves as terms \( E \) except that each execution \( a \) becomes \( \rho(a) \);
- \( "E_1 || E_2" \) 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.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>( a.E \rightarrow E )</td>
<td>if ( a \notin L )</td>
</tr>
<tr>
<td>2.</td>
<td>( E \rightarrow E' )</td>
<td>if ( a \in L )</td>
</tr>
<tr>
<td>3.</td>
<td>( E/L \rightarrow E'/L )</td>
<td>if ( a \in L )</td>
</tr>
<tr>
<td>4.</td>
<td>( E_1 \rightarrow E_1' )</td>
<td>if ( a \notin S )</td>
</tr>
<tr>
<td>5.</td>
<td>( E_1</td>
<td></td>
</tr>
<tr>
<td>6.</td>
<td>( E_1 + E_2 \rightarrow E_1'</td>
<td></td>
</tr>
<tr>
<td>7.</td>
<td>( E_1 \rightarrow E' )</td>
<td></td>
</tr>
<tr>
<td>8.</td>
<td>( E_1 + E_2 \rightarrow E' )</td>
<td></td>
</tr>
<tr>
<td>9.</td>
<td>( E[\sigma] \rightarrow E'[\sigma] )</td>
<td>if ( A \Delta E )</td>
</tr>
<tr>
<td>10.</td>
<td>( A \rightarrow E' )</td>
<td></td>
</tr>
</tbody>
</table>

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 $Ite \in I_C^p \cap Ite$, provides a set of operations that are called by other components;
- $I_C^r$ is a set of requested interfaces. For any $Ite \in I_C^r \cap Ite$, $Ite$ 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 $A_C^i$ respectively. These three parts are disjoint;
- $L_C$ is a set of connections between $C$ and other components. For any connection $l_i \in L_C$, $l_i = \langle Rite, Plte, Ins, PL > \rangle$ which denotes that $C$ is connected with some other component. In the tuple, $Rite \in I_C^p$ is one of $C$’s requested interfaces, $Plte$ is the corresponding interface provided by the external component $C$, and $Rite$ and $Plte$ are matched in syntax. Hence, $C$ is able to call $C_i$ through $Plte$. The item $Ins$ is an instance of $C_i$ and $PL$ is the location of $C_i$.
- $P_C$ denotes the behavioral protocol of $C$, which is formal defined by PA. $P_C$ is defined in such a tuple $(S_C, \Gamma_C)$, where $S_C$ a finite set of states. We use $s_{\text{fin}}$ and $s_{\text{init}}$ 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 $\Gamma_C = \{(s_i, a, s_j) | s_i, s_j \in S_C, a \in A_C, s_i \xrightarrow{a} s_j\}$. This paper supposes that $P_C$ is defined as preserving determinacy, which means that if $P_C \xrightarrow{a} s_i \mathbin{\boxtimes} P_C \xrightarrow{a} s_j \mathbin{\boxtimes} \text{then it holds } P_C \underbrace{_{\text{fin}}} \subseteq P_C \underbrace{_{\text{fin}}}$. $P_C$ also holds correctly-terminated, which is for any state $s' \in S_C$, whenever $s' \neq s_{\text{fin}}$, there exist a $\sigma \in A_C$, and $s' \equiv s_{\text{fin}}$. We also use $trace_{\text{fin}}(P_C)$ denote $trace_{\text{fin}}(P_C) = \{\sigma | \sigma \in A_C \text{ and } s_{\text{init}} \xrightarrow{\sigma} s_{\text{fin}}\}$.

For two components $C_i = (I_{C_i}^p, I_{C_i}^r, A_{C_i}, L_{C_i}, P_{C_i})$, and $C_j = (I_{C_j}^p, I_{C_j}^r, A_{C_j}, L_{C_j}, P_{C_j})$ connected with each other through an interface $Ite$ provided by $C_i$, we will analyze their interaction behavior, $P_{C_i \parallel M} P_{C_j}$. If $C_i$ and $C_j$ are connected through more than one interfaces, we use $\kappa(C_i, C_j)$ denote the set of these interfaces, and the actions in these interfaces are denoted as $\alpha(C_i, C_j) = \bigcup_{Ite \in \kappa(C_i, C_j)} Ite \in \kappa(C_i, C_j)$. Clearly, if two components are disconnected, it holds that $\kappa(C_i, C_j) = \phi$. During the interactions, $C_i$ and $C_j$ will asynchronously execute the actions belonging to $\alpha(C_i, C_j)$. Based on this, we present the notion of behavioral compatibility as following:

**Definition 7** For two components $C_i \in C_1, C_2, ..., C_n \{ C_i \} = (I_{C_i}^p, I_{C_i}^r, A_{C_i}, L_{C_i}, P_{C_i})$ is formally defined as follows:

A software system containing $n$ components $\{ C_1, C_2, ..., C_n \} \in \{ C_i \} = (I_{C_i}^p, I_{C_i}^r, A_{C_i}, L_{C_i}, P_{C_i})$ is formally defined as follows:

**Definition 8** A software system, $MCS$, containing $n$ component, is denoted as $MCS = \langle V_M, E_M, I_M, T_M, P_M \rangle$, where:

- $V_M = \{C_1, C_2, ..., C_n\}$ is a set of software components assembled in the system, and $C_i \in C_1, C_2, ..., C_n$.
- $E_M = \bigcup_{(C_i, C_j)} \kappa(C_i, C_j)$ 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 $I_M = I_M^p \cup I_M^r$, $I_M^p = \bigcup_{C_i \in E_M} I_M^p \cap \bigcup_{C_i \in E_M} T_M^p \cap \bigcup_{C_i \in E_M} P_M^p$ is the set of interfaces provided by $MCS$, and $I_M^r = \bigcup_{C_i \in E_M} I_M^r \cap \bigcup_{C_i \in E_M} T_M^r \cap \bigcup_{C_i \in E_M} P_M^r$ is the set of interfaces requested by it;
- $T_M$ is a set of all connections in $MCS$, $T_M = \bigcup_{C_i \in E_M} T_M$;
- $P_M$ is the internal behavior of $MCS$ and it is defined as $P_M = P_{C_1} \parallel (P_{C_2} \parallel \cdots \parallel P_{C_n}) P_{C_1} \parallel (P_{C_2} \parallel \cdots \parallel P_{C_n})$.
The state \( s_m \) of \( P_m \) is denoted as \( s_m = (s_1, \ldots, s_n) \) where \( s_i \in S_C \), \( 1 \leq i \leq n \). 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 \( a \in A_C^n \) and such a state transition \( (s_n, a, s'_n) \in \Gamma_C \), and for the states of the other component \( C_j \) (1 \( \leq j \leq n, j \neq i \)), it holds \( s_j = s'_j \);
- there are an action \( a \in Ite \), \( Ite \in e(C_i, C_j) \) (1 \( \leq i \), \( j \leq n, i \neq j \)), and two state transitions \( (s_n, a, s'_n) \in \Gamma_C \) and \( (s_n, a, s'_n) \in \Gamma_C \), and for the states of the other components \( C_k \) (1 \( \leq k \leq n, k \neq i, j \)), 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, \( \{C_1, C_2, \ldots, C_n\} \) \((C_i \nless t^p_i, I^p_i, A_C, L_C, P_C) > 1\) \( \), its behavior \( P_m \) is deadlock-free, if and only if, for \( P_m \)'s initial state \((s^p_1, s^p_2, \ldots, s^p_n) \), there exists a trace \( \sigma \in (\bigcup_{i=1}^{n} A_C) \) with \( \sigma \upharpoonright \bigcup_{i=1}^{n}(A_i, C_j) \) \( 
\)

and it holds that 

\[(s^p_1, s^p_2, \ldots, s^p_n) \Rightarrow (s^p_1, s^p_2, \ldots, s^p_n).\]

For a software components system MCS holding a deadlock-free behavior, if one of its components \( C_j \) is replaced by a new one \( C'_j \), \( C_j \) still preserves behavioral compatibility with other components in the system, we say that \( C_j \) can be replaced by \( C'_j \). Suppose a component \( C_i \) \( \nless \) \( t^p_i, I^p_i, A_C, L_C, P_C \) will be replaced by a new one \( C_i \) \( \nless \) \( t^p_i, I^p_i, A_C, L_C, P_C \). To meet the requirement of replacement, a necessary condition must be holding, which is \( t^p_i \subseteq I^p_i \cap A_C \subseteq A_C \). 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 \( C_j \) interacts with \( C_i \) through an interface \( Ite \) provided by \( C_j \). To preserve behavioral compatibility, \( C_j \) only concerns the behavior of \( C_i \) shown on \( Ite \). The same to \( C_i \), it only concerns whether the corresponding request behavior of \( C_j \) 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 \( C_i \) and \( C_j \) \( \nless \) \( \langle t^p_i, I^p_i, A_C, L_C, P_C \rangle > \) and \( C_j \) \( \nless \) \( \langle t^p_j, I^p_j, A_C, L_C, P_C \rangle > \) 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:

- \( (P_i \| P_j)/D1 = B P_i/D2 \) here \( D1 = A_C \cup A_C \) \( \nless \) \( Ite \) \( \nless \) \( D2 = A_C \nless \) \( Ite \)
- \( \text{trace}_{A_C}(P_i) \nless \text{trace}_{A_C}(P_j) \nless Ite \)

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 \( C_i = \langle t^p_i, I^p_i, A_C, L_C, P_C \rangle > \) and \( C_j = \langle t^p_j, I^p_j, A_C, L_C, P_C \rangle > \) interact with each other through two interfaces \( Ite_1 \) and \( Ite_2 \). \( Ite_1 \) \( \nless t^p_i \cap I^p_i \wedge Ite_2 \) \( \nless t^p_j \cap I^p_j \), 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:

\[ \text{trace}_{A_C}(P_i) \nless \text{trace}_{A_C}(P_j) \nless Ite_1 \]
The additional operations in the executions of other operations. The execution of these new operations will not influence the behavior of \(P_c\), holds that (1) \(A_p = A_p \sqcup A_p = A_p \sqcup A_p = A_p\); (2) \(I_t = I_t \sqcup I_t = I_t\); (3) \(P_C = P_C\).

From the definition, we can see that the behavioral protocol of \(C\) 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 \(C\), and \(C\) interact through an interface \(Ite\) provided by \(C\). Their behavioral compatibility satisfies the conditions presented in theorem 1. Now, suppose \(C\) is the dual component of \(C\), \(P_C\), \(P_C\) can simulate \(P_C\), where \(D = A_p = -Ite\).

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

**Theorem 5** Suppose two components \(C\), and \(C\) interact through an interface \(Ite\) provided by \(C\). Their behavioral compatibility follow the conditions presented in theorem 1. Now, a new component \(C\) will be updated by a new one \(C\), \(P_C\), \(P_C\) is the dual of \(C\), and \(P_C\) has a provided interface \(Ite\) corresponding to \(Ite\). If \(P_C\), \(P_C\) can be behaviorally substituted by \(C\).

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 \(C\) is a dual component of \(C\), if the following conditions hold (1) \(A_p = A_p \sqcup A_p = A_p \sqcup A_p = A_p\); (2) \(I_t = I_t \sqcup I_t = I_t\); (3) \(P_C = P_C\).

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

4.2 Behavioral Substitutability in a Two-Component Assembly

Suppose two components \(C\), and \(C\) 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 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 \(C\), and \(C\) are assembled through an interface \(Ite\) provided by \(C\). Their behavioral compatibility follow the conditions presented in theorem 1. Now, \(C\) will be updated by a new one \(C\) with a provided interface \(Ite\) corresponding to \(Ite\), and \(Ite \subseteq Ite\). If it holds that (1) \((P_C \land (Ite\land Ite))(Ite)\land (Ite)\) and \(P_C\), \(P_C\) can be behaviorally substituted by \(C\).

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, \(Ite\). 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 \(Ite \sqcup Ite = \phi\). It may just extend the provided behavior on the interface. However, theorem 3 can’t verify this case.
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 weaker 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 \(C_i = \langle I^p_{i}, I^b_{i}, A_{i}, L_{i}, P_{i} \rangle >\) and \(C_j = \langle I^p_{j}, I^b_{j}, A_{j}, L_{j}, P_{j} \rangle >\) interact with each other through two interfaces \(Ite_1 \) and \(Ite_2\), \(Ite_1 \in I^p_{i} \cap I^p_{j}\) and \(Ite_2 \in I^b_{i} \cap I^b_{j}\). Their behavioral compatibility satisfies the conditions presented in theorem 2. Now, a new component \(C_j’= \langle I^p_{j’}, I^b_{j’}, A_{j’}, L_{j’}, P_{j’} \rangle >\) is used to replace \(C_j\). Its new provided interface is \(Ite_1’\) corresponding to \(Ite_2\), with \(Ite_2 \subseteq Ite_1’\), and its requested interface is still \(Ite_1\). Let \(C_{ij}\) be the dual component of component \(C_j\), if \(P_{C_i} \) and \(P_{C_j} \) satisfy the following conditions:

- \(P_{C_j} \sqsubset S_{i,j} = P_{C_i} \sqcup (A_{i} \sqcap \neg Ite_1’)\) and \(trace_{sa}(P_{C_j}) \uparrow Ite_1’ \subseteq trace_{sa}(P_{C_i}) \uparrow Ite_1’\)
- \(P_{C_j} \sqsubset S_{i,j} = P_{C_i} \sqcup (A_{i} \sqcap \neg Ite_1)\) and \(trace_{sa}(P_{C_j}) \uparrow Ite_1 \subseteq trace_{sa}(P_{C_i}) \uparrow Ite_1\)

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 \(P_M = P_C \sqcup \ldots \sqcup P_{C_j} \sqcup P_{C_{j+1}} \sqcup \ldots \sqcup P_{C_n}\) denote the behavior of the system. Clearly, \(P_M\) can also be defined in such a form, \(P_M = P_{C_1} \sqcup \ldots \sqcup P' \sqcup P_{C_2} \sqcup \ldots \sqcup P_{C_n}\), where \(S_i = \bigcup_{i \in \text{Component}} (A_i, C_i)\) and \(P' = P_{C_1} \sqcup \ldots \sqcup P_{C_{j-1}} \sqcup P_{C_{j+1}} \sqcup \ldots \sqcup P_{C_n}\). 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, \(\{C_1, C_2, \ldots, C_n\}\) with \(P_{C_1} \sqsubset \ldots \sqsubset P_{C_n}\) and its behavior \(P_M\) be \(P_{C_1} \sqcup \ldots \sqcup P_{C_n}\). There is a component \(C_i\) in MCS with \(I^p_{i} = \phi\), and its behavior \(P_{C_i}\) satisfying the following conditions: 1) \(P_{C_i} \sqsubseteq P' \sqcup D\), where \(D = \bigcup_{i \in \text{Component}} (A_i, \neg S_i)\) and \(P' = \bigcup_{i \in \text{Component}} (\neg S_i)\); and 2) \(trace_{sa}(P') \uparrow S_i \subseteq trace_{sa}(P_{C_i}) \uparrow S_i\). Now, a new component \(C_i’= \langle I^p_{i’}, I^b_{i’}, A_{i’}, L_{i’}, P_{i’} \rangle \) is used to replace \(C_i\), and it holds that \(I^p_{i’} \subseteq I^p_{i}\) and \(A_{i’} \subseteq A_{i}\). Set \(S_{i’} = \bigcup_{i \in \text{Component}} (A_i, \neg S_{i’})\) and \(C_{i’}\) be the dual component of \(C_i\). If \(P_{C_i} \) and \(P_{C_i’} \) satisfy the following conditions:

- \(P_{C_i} \sqsubseteq D1 = P_{C_i} \sqcup \ldots \sqcup P_{C_{j-1}} \sqcup D2\), where \(D1 = A_{i’} \sqcup A_{i}\), \(D2 = A_{i’} \sqcup \neg S_{i’}\);
- \(trace_{sa}(P_{C_i’}) \uparrow S_{i’} \subseteq trace_{sa}(P_{C_i}) \uparrow S_{i’}\);

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, \( \{ C_k, C_2, ..., C_n \} \) \( \subseteq C \), and its behavior \( P_m \) be \( P_m = \bigcirc_{i \in C_k} (P_1 \cap C_i) \cap C_j \cap C_{ij} P_3 \cap C_j \cap C_{ij} P_3 \cap \bigcirc_{i \in C_k} \bigcirc_{j \in C_n} \bigcirc_{l \in C_{ij}} P_4 \). There is a component \( C_i \in \text{MCS} \) with \( I_1^i \neq \phi \cap I_2^i \neq \phi \). Let \( S_i = \bigcup_{I \in C_k} (I \cap I_i) \), and \( S_i^j = S_i \setminus \bigcup_{I \in C_k} (I \cap I_i) \) \( \cap \bigcup_{I \in C_n} (I \cap I_i) \), the behavior of \( C_i \), \( P_i \), meets the following conditions: 1) \( ( P_i \cap C_i \cap C_j ) \cap C_k \cap C_{ij} \cap C_{ij} P_3 \cap C_j \cap C_{ij} P_3 \cap C_j \cap C_{ij} P_3 \setminus \bigcup_{I \in C_k} (I \cap I_i) \), and \( \text{trace}_{C_i} (P_i^\top) S_i^j \subseteq \text{trace}_{C_i} (P_i^\top) S_i^j \); 2) \( ( P_i \cap C_i \cap C_j ) \cap C_k \cap C_{ij} \cap C_{ij} P_3 \cap C_j \cap C_{ij} P_3 \setminus \bigcup_{I \in C_k} (I \cap I_i) \), and \( \text{trace}_{C_i} (P_i^\top) S_i^j \subseteq \text{trace}_{C_i} (P_i^\top) S_i^j \). 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, \( \text{BookShop} (C_{kb}), \text{BookShop} (C_{bb}) \) and \( \text{Bank} (C_{ba}) \). In the system, the component \( \text{BookShop} \) registers at \( \text{BookBroker} \) first. When a user wants to buy books, it will call the interface operation \( \text{getABook} \) provided by component \( \text{BookBroker} \). \( \text{BookBroker} \) inquires \( \text{BookShop} \) whether there is the book in stock by calling the interface \( \text{inStock} \) provided by \( \text{BookShop} \). If \( \text{BookShop} \) has the book, \( \text{BookBroker} \) will order the book by calling the operation \( \text{Order} \), allow \( \text{BookShop} \) deliver books to \( \text{User} \) by calling the operation \( \text{deliver} \), and deposit money in \( \text{BookShop} \)'s bank account by calling the operation \( \text{deposit} \) provided by \( \text{Bank} \). Their assembly structure, interfaces described in IDLs and definitions are given in Figure 1~3. Obviously, the behavioral compatibility between \( C_{ba} \) and \( C_{bb} \) on interfaces \( \text{Ite}_{\text{BookShop}} \) and \( \text{Ite}_{\text{BookShop}} \) satisfies the conditions in theorem 2, and the behavioral compatibility between \( C_{ba} \) and \( C_{bb} \) on interfaces \( \text{Ite}_{\text{BookAccount}} \) satisfies the conditions presented in theorem 1.

Now, the system is upgraded to provide more functions. Two new components \( \text{Bank} \) and \( \text{Bookshop} \) 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 \( \text{Bank} \) add a new operation \( \text{Query} \) in its provided interface, which will be used to query the client’s deposit. Component \( \text{Bookshop} \) add a new function to cancel the orders of books in its interface, \( \text{cancelOrder} \). Their behavioral is also adapted. Now we will verify whether the new components \( \text{Bank} \) and \( \text{Bookshop} \) can replace the old ones successfully. In the upgraded system, component \( \text{Bank} \) only interacts with \( \text{BookBroker} \) 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 \( ( P_{c_{bb}} \setminus D1/D2 \neq 0 \) \( P_{c_{ba}} / D3 \), where \( D1=\text{Ite}_{\text{BookAccount}} \setminus \text{Ite}_{\text{BookAccount}} = \{ \text{query}, \text{query}_r \} \), \( D2= \text{A}_{c_{ba}} \) \( \setminus \text{Ite}_{\text{BookAccount}} \) \( = \phi \) and \( D3= \text{A}_{c_{bb}} \) \( \setminus \text{Ite}_{\text{BookAccount}} \) \( = \phi \). It meets the conditions presented in theorem 3. On the other hand, component \( \text{Bookshop} \) only interacts with \( \text{BookBroker} \) through two interfaces. We will use the theorem 6 to verify the behavioral compatibility under this scenario.
An interface provided by BookShop: Book_Shop
interface Book_Shop{
  struct BookRef { string ISBN, float price; }
  boolean inStock(string title, in string author);
  void order( in BookRef b, out account a, out string purchaseID);
  date deliver( in string purchaseID, in string receipt, string address);
};

An interface provided by BookBroker: Broker_Shop
interface Broker_Shop{
  void register( in Bookshop b);
  void unregister( in Bookshop b);
};

An interface provided by BookBroker: Broker_User
interface Broker_User{
  boolean getABook( in string author, in string title,
                   out account a, out string purchaseID, out date purchase);
  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 receipt, string address);
};

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();
};

Fig.2 IDL interfaces provided by components Bank, BookShop and BookBroker
Clear, we can see that the following two conditions hold: 1) \( \langle P_{c_{as}} \| Ite1 \|= Ite2; P_{c_{as}} \rangle /D3 \models B_{c_{as}} / D4 \), where \( Ite1 = Ite_{Broker\_Shop} \), \( Ite2 = Ite_{Book\_Shop} \), \( D3 = A_{c_{as}} \cup A_{c_{as}} \) \( Ite_{Book\_Shop} = \{ \text{register, unregister} \} \), \( D4 = A_{c_{as}} \cup A_{c_{as}} \) \( Ite_{Book\_Shop} = \{ \text{register, unregister} \} \), and \( \text{trace}_{c_{as}}( P_{c_{as}} ) \| Ite2 \subseteq \text{trace}_{c_{as}}( P_{c_{as}} ) \| Ite2 \); 2) \( \langle P_{c_{as}} \| Ite1 \|= Ite2; P_{c_{as}} \rangle /D5 \models B_{c_{as}} / D6 \), where \( D5 = A_{c_{as}} \cup A_{c_{as}} \) \( Ite_{Broker\_Shop} = \{ \text{inStock, inStock\_r, order, deliver, cancelOrder, cancelOrder\_r, deliver\_r} \} \), \( D6 = A_{c_{as}} \cup A_{c_{as}} \) \( Ite_{Broker\_Shop} = \{ \text{inStock, inStock\_r, order, deliver, cancelOrder, cancelOrder\_r, deliver\_r} \} \), \( \text{trace}_{c_{as}}( P_{c_{as}} ) \| Ite1 \subseteq \text{trace}_{c_{as}}( P_{c_{as}} ) \| Ite1 \). 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 \( \pi \) calculus, but they didn’t take 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. 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. 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. 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. 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. 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. 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. 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: