Journal of Computer and Communications, 2015, 3, 44-53
Published Online May 2015 in SciRes. http://www.scirp.org/journal/jcc
http://dx.doi.org/10.4236/jcc.2015.35006

Software Components Behavioral Adaptation Based on LTS

Chi Zhang

School of Software and Communication Engineering, Jiangxi University of Finance and Economics, Nanchang, China
Email: zhangchi_1972@163.com

Received March 2015

Abstract

Compatibility is the precondition to ensure the correct interaction among components in composition process, how to make the mismatch components coordinate correctly is a vital problem in component composition. This paper first modeled component behavior by LTS and expressed action mapping as synchronous vector then defined the sequential relationship among synchronous vectors as adaptation contract. Thirdly we analyzed the different mismatch situations and corresponding adaptation strategies. At last designed adaptation algorithm to produce adaptor specification automatically and ensured the mismatch components can correct interaction under the mediation of adaptor and verified the validity of proposed method through an application system.

Keywords

Component, Behavioral Protocol, Synchronous Vector, Adaptation Contract, LTS

1. Introduction

Component based software development (CBSD) that aims at constructing new systems by compositing prefabricated reusable components have been tested can shorten the development cycle and improve software quality [1]. Composition is the core of component model [2]. Only components participating in the assembly be fully compatible can ensure correct interaction among components, however, usually components are provided by different software vendors so that the difference of operations name, operations order and operations granularity that are defined in component interfaces is almost inevitable due to different use perspectives. Incompatibility among components caused by these differences during the assembly process is almost inevitable too. Whether the incompatible assembly can be adaptable and, if adaptable, how to generate the adaptor specification are important problems that must be solved in the assembly practice.

Interface should contain full specification needed by interaction because Interfaces are the only interaction channels between components. At present mainstream component techniques (such as CCM, DCOM and EJB) just define syntax layer interaction information through Interface Description Language (IDL) thus the interface shows as a series of discrete operations set, however, the sequential relationship of these operations not defined clearly so can't inspect the behavioral characteristics of component interaction [3]-[5]. Labeled Transition Sys-

How to cite this paper: Zhang, C. (2015) Software Components Behavioral Adaptation Based on LTS. *Journal of Computer and Communications*, **3**, 44-53. http://dx.doi.org/10.4236/jcc.2015.35006

tem [3] (LTS) not only has graphical property but also has strict mathematic foundation so be selected to model component behavioral. Thereafter, we using synchronous product to stimulate the interaction relationship of receiving and sending messages, and then check whether there arise the deadlock in synchronous product. The interaction process needs adaption if deadlock exists in synchronous product. The first step of adaption process is to define the corresponding actions between sender and receiver as synchronous vectors. Then, define the sequential relationship of synchronous vectors as adaptation contract. In the end, analyze different mismatch types and adaptation strategies and design the adaptation algorithm to build adaptor specification automatically thus to ensure the correct interaction between the components with mismatch behavioral.

The remainder of the paper is organized as follows: Section 2 formally introduces our component model and the case study we use throughout this paper. Section 3 presents how to judge the behavioral compatibility and analyzes different mismatch situations and corresponding adaptation strategies. Section 4 defines the synchronous vectors according to the messages receiving and sending relationship and defines the sequential relationship of synchronous vectors as adaptation contract then designs the adaptation algorithm to work out the mismatch interaction behavioral. Section 5 compares our approach with related works. Section 6 ends the paper with some concluding remarks.

2. Component Behavioral Model Based on LTS

Definition 1 (Component View) Component view CV can be defined as a triple tuple $CV = (I^p, I^R, P)$, where:

I^p is provided interface, including operations can be called by other components, namely $I^p = \{a_1, a_2, \ldots, a_n\}$

I^R is require interface, including operations request to other components, namely $I^R = \{a_1, a_2, \ldots, a_m\}$

P is behavioral protocol of the component and is formally defined through LTS, P describes the specification and constraint of the component's behavioral protocol and is formally defined as a quintuple $P = \{A, S, I, F, T\}$, where:

① A is a set of events;

② S is a set of states;

③ I is initial state, $I \in S$;

④ F is final states, $F \subseteq S$;

⑤ $T \subseteq S \times A \times S$, is a set of finite state transition in component interaction;

Component behavioral is presented by LTS, Message-based communication between components is therefore represented using events relative to the emission (denoted by!, defined as dir (a) =?) and reception (denoted by?, defined as dir (a) =!) of messages corresponding to operation calls. Complementary events are denoted with the same name of message and opposite directions, namely $\overline{a?} = a!; \overline{a!} = a?$.

The **Figure 1** shows two components, customer and server, of a Video on Demand (VoD) system. The graph (a) is customer component LTS, where customers can play as two roles: guest and user. The role of guest is able to inquire (*Query!*) the list (*List?*) of videos and choice (*Choice!*) one of them. If guest isn't interested in any of the videos, he or she can execute (*Shutdown!*) to exit the system. The other role user is required to log into the

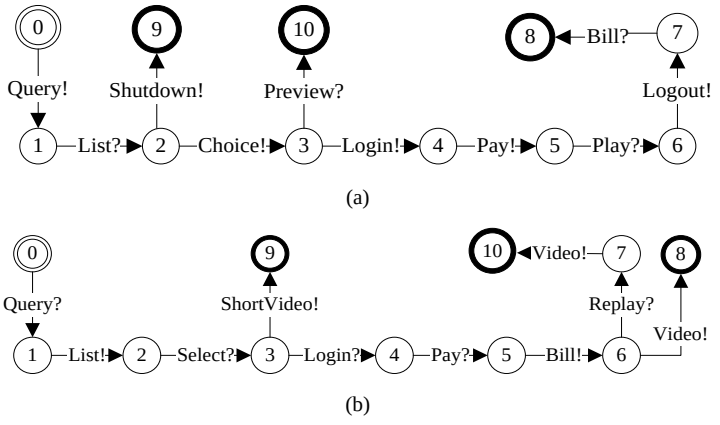

Figure 1. (a) Customer component LTS and (b) VoD server LTS in the VoD system.

system (*Login!*) before playing any video and input the form of payment (*Pay*?). After playing (*Play*?) the user logging out the system (*Logout!*) and get the details of the bill (*Bill*?). The graph (b) is the server LTS in the system. It receives user' inquiry (*Query*?) and provides the querying results in list (*List!*) where customer can select (*Select*?) one video. If customer is user role the server requires he or she log in (*Login*?) first and provide form of payment (*Pay*?). Then, the details of bill information will be sent to user before the video playing (*Video!*). After paying, the user can choose replay (*Replay*?) the video (*Video!*), If the customer is guest role he or she only be provided free short video preview (*ShortVideo!*). The LTS model can describe not only the sending and receiving messages between components but also sequential relationship of these messages.

3. Judgment of Composition Compatibility and Analysis of Mismatch Behavioral

3.1. Judgment of Composition Compatibility

As far as the application system composited by several components is concerned, the running process of the system is presented as a series of operations invocation among these components. In the system, synchronous transition happens to at least two components under the effect of complementary events. It means that when one component sending the message a, there is another component is ready and able to receive the message a. The system can interact correctly and terminated must meet two conditions [6] [7]: (1) Synchronous transition happens anytime within the system. (2) Every component is able to transit from its initial state to its terminal state.

In order to simulate the system running process and inspect the system behavioral compatibility, we define the synchronous product, contains all of the interaction behavioral among components, under the effect of complementary event (a, a), and synchronous product still is a LTS [8].

Definition 2 (Synchronous Product) The synchronous product of several components LTS $C_i = (A_i, S_i, I_i, F_i, T_i)$, $i \in \{1,..., n\}$, is LTS $C_1 \| ... \| C_n = (A, S, I, F, T)$, where

① $A = A_1 \cup \{\varepsilon\} \times ... \times A_n \cup \{\varepsilon\}$;

②$S = S_1 \times ... \times S_n$;

③$I = (I_1,..., I_n)$;

④$F = (F_1,..., F_n)$

⑤$T$is defined by following rules:

$\forall (s_1,..., s_n) \in S$, $\forall i, j \in \{1,..., n\}$, $i < j$; if $\exists (s_i, a, s_i') \in T_i$, $\exists (s_j, \overline{a}, s_j') \in T_j$, $(x_1,..., x_n) \in S$, $((s_1,..., s_n), (l_1,..., l_n), (x_1,..., x_n)) \in T$, $\forall k \in \{1,..., n\}$:

If $k = i$, then $l_k = a$, $x_k = s_i'$;

If $k = j$, then $l_k = \overline{a}$, $x_k = s_j'$;

If $k \neq i, j$, then $l_k = \varepsilon$, $x_k = s_k$;

According to definition 2, the necessary and sufficient condition for further transition within one of the composite state in the synchronous product is that there exist component C_i and C_j, they execute the complementary events at the same time and they transit from state s_i and s_j in their own LTS respectively. The other components don't make any transition. (Denoted by ε); as far as one component is concerned if it is in a nonterminal state and can't transit further then we define this state as deadlock state.

Definition 3 (Deadlock State) Let C be a LTS = (A, S, I, F, T), a state is deadlock state, noted by $dead(s)$, if it is in S, not in F, and has no outgoing transition: $s \in S \wedge s \neq F \wedge (\neg\exists l \in A, s' \in S, (s, l, s') \in T)$.

Due to the system that formed by several components receiving and sending messages synchronously is also a LTS, the compatibility judgment can be done through inspecting whether there is a deadlock in the synchronous product LTS.

Definition 4 (Deadlock) if there is a state s in S such that $dead(s)$, then $LTS = (A, S, I, F, T)$is deadlock.

We can design algorithm to check system compatibility automatically according to definition 3 and 4. The input of algorithm is synchronous product LTS and output is True or False, if the output is False then there has deadlock in synchronous product LTS and system incompatible else system compatible.

Algorithm 1: Check compatibility

Input: synchronous product LTS = (A, S, I, F, T)

Output: True or False

1: $s = I$;

2: if $(\neg\exists l \in A, s' \in S, (s, l, s') \in T)$ then output (False)

3: else While (true)

4: {*s*' = subsequence (*s*);
5: *s* = *s*';
6: If ($s \notin F \wedge \ (\neg\exists\, l \in A, s' \in S, (s, l, s') \in T)$)
7: Then {output (False);break;}
8: Else if ($s \in F$)
9: Then {output (True);break} }

The algorithm 1 is designed to inspect whether the initial node in the synchronous product can transit or not at first. If there is a deadlock (sentence 1 - 2) then exit system, otherwise, check whether the successor node can transit or not if no then exit system (sentences 3 - 7). If it can transit outgoing to the terminal state, it indicates that there not exists any deadlock in the system (sentences 8 - 9).

Regardless of looping, the synchronous product forms a directed acyclic graph and we can use depth first traversal to inspect whether the transition is executable in each node. The algorithm complexity is O (|S|) in which the *S* is the state number of synchronous product.

Now we can check the compatibility between customer component and VoD server component in **Figure 1**. Two components transit to state (②, ②)respectively from their initial status under the effect of complementary events (*Query*!, *Query*?) and (*List*?, *List*!). After that, even if the direction of the event (*Choice*!, *Select*?) is reverse, but the name of the messages are not the same, so, they cannot synchronize and the state (②, ②) isn't the terminal state, which leads to deadlock.

3.2. Mismatch Analysis and Adaptation Strategies

In the practice of composition, components are usually developed by different third vendors. Since the usage exception of the components is not exactly the same, the operation name, operation sequence and operation granularity of the components inevitably be defined differently. It results in the situation of incompatibility when composition. And meantime, for the system integrators, it is impossible to modify the codes of the components due to their black-box characters; the only way they can use is non-intrusive adaptation mechanism to eliminate the incompatibility among components.

The behavior of each component forms a directed acyclic graph. For one component, each time of its execution corresponding a path from initial state to terminal state. The definition 5 defines the concept of interaction path.

Definition 5 (Interaction Path)In one execution, the sending/receiving message sequence of component C_i when it interacting with other components, in the LTS model, its corresponding a transition sequence from initial state to terminal state, which is expressed as $P_i = t_{i1}, t_{i2}, \ldots, t_{in}$ and corresponding massage sequence is $MS_i = l_{i1}, l_{i2}, \ldots, l_{in}$.

This paper assumes that the operation granularity is the same. Namely, one operation of a component not have several counterpoints in another component, componentsC_1 and C_2 will meet the four mismatch cases listed below:

(1) Name mismatch (1 - 1): The operation functions defined by both of the receiving and sending components are same but their names are different, namely: $l_{1i} \in MS \wedge l_{2j} \in MS_2 \wedge \ (l_{1i}, l_{2j}) \in V \wedge$ (name (l_{1i}) ≠ name (l_{2j})), $i, j \in [1, n]$. Like the message pair (*Choice*!, *Select*?) in the VoD system. It is also found that there are message pairs (*ShortVideo*!, *Preview*?) and (*Play*?, *Video*!), which have different names but same function. Adaptor just needs converting the names.

(2) Unanticipated reception (1-0): The sender send a message, but the receiver didn't define any corresponding operation, namely: $l_{1i} \in MS_1 \wedge$ (dir (l_{1i}) =!) $\wedge \ (l_{1i}, \varepsilon) \in V, i \in [1, n]$. For the VoD system the customer sends the messages of shutting down (Shutdown!) and logout (Logout!), but there isn't any corresponding operation defined in the server. This type of message indicates the independence evolutions of single component, as for as adaptation is concerned, these messages are filtered instead of sending to the receiver.

(3) Expected reception undefined (0 - 1): The receiver expects to receive certain message, but the sender is not able to send the corresponding message, namely: $l_{1i} \in MS_1 \wedge$ (dir (l_{1i}) =?) $\wedge \ (l_{1i}, \varepsilon) \in V, i \in [1, n]$. In out demonstration VoD system the server component expects to receive the *Replay* call, but the customer either didn't or won't call this operation. The receiver lies in the deadlock state in this interaction path and cannot be adapted. As for as adaptation is concerned, transitions and states after deadlock state of the receiver be deleted from its LTS because this path is inadaptable.

(4) Misorder (m-m): Both of the sender and the receiver define corresponding operations, but the order of these operations isn't the same, namely: $l_{1i} \in MS_1 \wedge l_{2j} \in MS_2 \wedge (l_{1i}, l_{2j}) \in V \wedge (i \neq j)$, $i, j \in [1, n]$. For the VoD system, the sequence of operations for the user role is: Login (*Login*!) $\rightarrow$ Pay (*Pay*!) $\rightarrow$ Play (*Play*?) $\rightarrow$ Bill (*Bill*?), while, for the server, the sequence of operations is: Login (*Login*?) $\rightarrow$ Pay (*Pay*!) $\rightarrow$ Bill (*Bill*?) $\rightarrow$ Play (*Play*?). As for adaptation process, to the adaptor saving the message sent by sender; to the receiver, checking whether the corresponding message that the receiver needs to receive has been saved. If so, then adaptor forwards it. Else waiting for another one until all the messages that ought to be received are being forwarded to the receiver. Otherwise, the adaptation fails.

As for the software system composited by components, components cannot be assembled directly if mismatching. Therefore, introducing an adaptor to mediate the interaction between components with mismatch behavioral and ensure the incompatible components interact correctly. If exists such an adaptor, it's adaptable among those components.

4. Generation of Adaptation Specification

According to the aforementioned analysis, the adaptor is an independent component, which is able to partly or wholly resolve the incompatibility problems among components. The functions of the adaptor include [9]: (1) receive messages sent by sender; (2) cache the received messages; (3) send the cached message to receiver; (4) filter the unanticipated reception messages

Definition 6 (Adaptor specification) An adaptor specification of a set of components $C_i = \{A_i, S_i, I_i, F_i, T_i\}$, $i \in [1, n]$, is a LTS = $\{A_A, S_A, I_A, F_A, T_A\}$, where: S_A is a set of adaptor state, $S_A \subseteq S_1 \times \ldots \times S_n$, $A_A \subseteq A_1 \cup \ldots \cup A_n$ is the set of adaptor actions, $I_A = I_1 \times \ldots \times I_n$ is initial state of adaptor, $F_A \subseteq F_1 \times \ldots \times F_n$ is a set of terminal states, $T_A \subseteq S_A \times A_A \times S_A$ is a set of transition.

4.1. Adaptation Contract

In order to express the corresponding relationship of messages during synchronous interaction among components, the definition 7 defines the concepts of synchronous vector. Each event in synchronous vector is executed by one component and each vector represents the corresponding relationship among the messages that sent and received in their synchronous interaction.

Definition 7 (Synchronous Vector) A synchronous vector V for a set of components $C_i = \{A_i, S_i, I_i, F_i, T_i\}$, $i \in [1, n]$, is a tuple $< C_1{:}l_1;\ldots;C_i{:}l_i;\ldots;C_n{:}l_n >$, where C_i is component identifier, l_i is the event component executed, $l_i \in (A_i \cup \{\varepsilon\})$, ε meaning that a component does not participate in a synchronization.

For VoD system, the synchronous vectors between customer and VoD server can be indicated according to the corresponding relation of their events.

$V_{Query} \leq$ Client: Query!, Server: Query? > ;
$V_{list} \leq$ Client: List?, Server: List! > ;
$V_{Select} \leq$ Client: Choice!, Server: Select? > ;
$V_{Login} \leq$ Client: Login!, Server: Login? > ;
$V_{Pay} \leq$ Client: Pay!, Server: Pay? > ;
$V_{Logout} \leq$ Client: Logout!, Server: ε > ;
$V_{Bill} \leq$ Client: Bill?, Server: Bill! > ;
$V_{Play} \leq$ Client: Play?, Server: Video! > ;
$V_{Replay} \leq$ Client: ε, Server: Replay? > ;
$V_{Shutdown} \leq$ Client: Shutdown!, Server: ε > ;
$V_{Preview} \leq$ Client: Preview?, Server: shortvideo! >;

Synchronous vector only express the corresponding relation of receive/send messages among components. In order to describe the messages interaction relation among components from global perseptive, definition 8 still use LTS to express the sequence of synchronous vectors.

Definition 8 (Vector LTS) A vector LTS for a set of vectors is an LTS $L = (V_L, S_L, I_L, F_L, T_L)$, where labels are vectors.

Component behavior LTS shows the operation call relation in one component. Synchronous vector shows the corresponding relation between messages sent and received among interaction components. Vector LTS indicates the sequential of Synchronous vectors. As for the adaptation of interaction, three of the relations need to be

met simultaneously, namely adaptation contract.

Definition 9 (Adaptation Contract) An adaptation contract of a set of components $C_i = \{A_i, S_i, I_i, F_i, T_i\}$, $i \in [1, n]$, is a couple (V, L), where V is a set of vectors for component C_i and $L$is a vector LTS for V.

For our demonstration VoD system the role of customers can be classified as two roles: Guest and User. Under the guest role, it's available to inquire and get free short videos. Under the user role, it's available for him or her to inquire video, pay for it and then play. The **Figure 2** shows adaptation contracts for two different roles. The letter G represents the adaptation contract for guest and the letter U represents the adaptation contract of user.

On the basis of forming the adaptation contract, the states of adaptation contract can be regarded as nodes, the transition under the effect of synchronous vector can be seen as edges, and the adaptation contract can be treated as a directed acyclic graph. For this graph, we can traversal each path based on depth-first search algorithm, during this process, synchronous vector of every transition shows the relation of receiving and sending messages under current state. In one path, if there is a node that receiver didn't receive the expected message in the end then this node is deadlock state and this path inadaptable. Therefore, all the states following this deadlock state and outgoing transitions from these states should be deleted from adaptation contract. In the **Figure 2**, there isn't any send message that match the receive message in the vector $V_{Replay} \leq \varepsilon$, *Replay*? >, so the path this vector lies in is incompatible. We can get new adaptation contract of the VoD system, showed in **Figure 3**, by remove states U4, U5, U7 and their corresponding transitions in adaptation contract showed in **Figure 2**.

4.2. Generation of Adaptation Specification

After getting the adaptation contract, from the point of view of adaptor, in any point of time, the send and receive relation must be one of the four situations listed below:

(1) The sender and receiver neither receiving nor sending messages: It shows that two sides of interaction are in their terminal states.

(2) One side sending message and another side receiving message: If the message sending by sender is exactly what receiver expects to receive, the adaptor will forward the expected message to receiver directly. If the sending message isn't the expected one by another side, the adaptor will receive this message and cache it, the sender continues to transit, while the state of receiver doesn't update.

(3) Receiving messages simultaneously: Inspect whether the expected message is already cached in adaptor, if it is, get the message out from adaptor, receiver and sender continues to transit. If the message just expected by one side is cached in adaptor, then this side transit further and the other side, which doesn't have matching message doesn't transit. If neither of two sides has their expected messages in cache, then the deadlock happens. The state of both sides doesn't change and the adaptation fails.

(4) Sending messages simultaneously: adaptor caches the received messages and both of sender and receiver transit further.

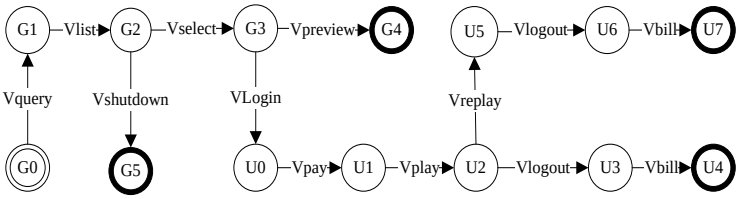

Figure 2. The adaptation contract of the VoD system.

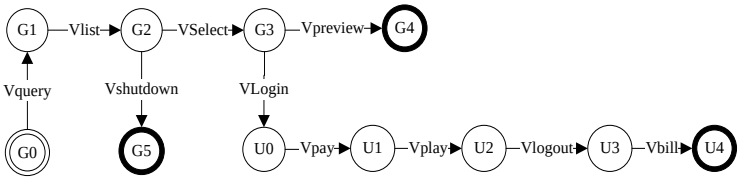

Figure 3. The adaptation contract of the VoD system after deleting the inadaptable path.

On the basis of above situations, algorithm can be designed and the adaptor specification can be generated automatically. For express convenience, we define several symbols: Em (s) = {l| (s, l!, s') $\in T \wedge s$, s' $\in S$} represents the set of messages that can be sent from state s. Rec (s) = {l| (s, l?, s') $\in T \wedge s$, s' $\in S$} represents the set of messages which can received from state s. Q is current set of messages in adaptor's cache. S_A is the state set of the adaptor and T_A is the transition set of the adaptor. In initial state: $Q = \varnothing$, $S_A = \varnothing$, $T_A = \varnothing$.

Algorithm 2: Generation of adaptation specification

Input: A set of components LTS $C_i = (A_i, S_i, I_i, F_i, T_i)$ and adaptation contract $LTS_{Ac} = (V, L)$;

Output: If adaptable then adaptable (C_i) = True and output the adaptor specification Ad = (A_A, S_A, I_A, F_A, T_A), else inadaptable and output adaptable (C_i) = False.

1: $LTS_{AC} = (V, L)$;
2: P_{AC} = removeun defined reception (LTS_{AC});
3: $S_A = S_{PAC}$; $T_A = \varnothing$; $Q = \varnothing$;adaptable (C_i) = True;
4: For all t = (s = (s_1, …, s_n), (l_1, …, l_n), s' = (s_1', …, s_n')) in P_{AC} do
5: for u = 1 to n
6: if $\in$ Em (s_u) Then
7: if (l_u, l_v) $\in V \wedge l_v \in$ Rec (s_v)then
8: t_u = (s_u, l_u!, s_u'); t_v = (s_v, l_v?, s_v'); $u = u + 1$;$v = v + 1$;
9: $T_A = T_A \cup \{(s, l_u, s''), (s'', {}^{l_v}, s''')\}$;
10: $S_A = S_A \cup \{s'', s'''\}$;
11: else if (l_u, l_v) $\notin V \wedge l_v \in$ Rec (s_v) then
12: if $l_v \in Q$ then
13: $Q = Q \cup l_u$; $Q = Q - l_v$;t_u = (s_u, l_u!, s_u');
t_v = (s_v, l_v?, s_v');$u = u + 1$;$v = v + 1$;
14: $T_A = T_A \cup \{(s, l_u, s''), (s', {}^{l_v}, s''')\}$;
15: $S_A = S_A \cup \{s'', s'''\}$;
16: else
17: $Q = Q \cup l_u$; t_u = (s_u, l_u!, s_u'); $u = u$+1;
18: $T_A = T_A \cup \{ (s, l_u, s'')\}$; $S_A = S_A \cup \{s''\}$;
19: end if
20: end if
21: end if
22: Else if $l_u \in$ Rec (s_u) $\wedge l_u \notin Q \neg\exists l_v \in$ Em (s_v) $\wedge$ v $\in$ [1, n] Then
23: adaptable (Ci) = False; exit;
24: end if
25: end for
26: end for
27: Return Ad = (A_{PAC}, S_A, I_{PAC}, F_{PAC}, T_A);

Algorithm removes the obvious inadaptable path that corresponds to the mismatch type expected reception undefined from adaptation contract firstly (sentence 2). The state information of adaptation contract be saved in adaptor specification and the transitions in adaptation contract will be replaced by receiving/sending messages. Initially, the set of messages cached by adaptor is empty and the adaptable label is true (sentence 3). Then for every transition of adaptation contract the algorithm inspects its enable Synchronous vector, if sending message exactly corresponds to the receiving message in the same vector then the component LTSs can transit, the adaptor only need to transfer the message to receiver and adds the transition and state information to adaptation contract (sentences 4 - 10). If the sending message doesn't be received then adaptor caches this message. If receiver exception message has been cached by adaptor then adaptor sends the message to receiver meanwhile adds transition and state to adaptation contract and both the sender and receiver can transit further (sentences 11 - 15). if the situation is no receiver accept the sending message while receiver exception message neither is current sending message nor be cached in adaptor then adaptor receives and caches the sending message and sender transits further but receiver state doesn't change. At the same time adaptor adds corresponding transition and state to adaptation contract (sentences 16 - 18). As far as the receiver is concerned, if there not exist send transition and the exception message has not been cached then adaptation fails and exit from system (sentences 22 - 23). If adaptable then in the end return the adaptor specification LTS (sentence 27).

Regardless of looping, component behavioral LTS and the adaptation contract form directed acyclic graphs, and algorithm use depth first traversals to inspect the transition in adaptation contract each component LTS simultaneously. The algorithm complexity is O ($|S|^{n+1}$), and the S is the maximum state number of component LTSs and adaptation contract LTS and n is number of components.

According to algorithm 2 the VoD system adaptor specification LST is generated automatically and showed in **Figure 4**.

Definition 10 (Correctness of adaptor specification)A set of components $C_i = \{A_i, S_i, I_i, F_i, T_i\}$, $i \in [1, n]$, under the mediated of adaptor specification Ad, every component can transit form initial state to terminal state, namely no deadlock state in Ad $\| C_1 \ldots \| C_n$.

To inspect the correctness of adaptation specification the synchronous product of components LTS $C_i = (A_i, S_i, I_i, F_i, T_i)$ and adaptor specification LTS Ad = $(A_{PAC}, S_A, I_{PAC}, F_{PAC}, T_A)$ needs to be calculated then judgments the compatibility by checking whether there is deadlock state in synchronous product according to algorithm 1. For the VoD system we can use the generated adaptor to eliminate the mismatch behavioral and ensure both of customer and server components transit from their initial states to terminal states.

5. Related Work

Adaptation technique has garnered wide attention in software engineering domain since Yellin and Storm firstly put forward in literature [10] that the adoption of adapting technique could resolve the incompatible problem among the interaction entities. At present, there are a lot of works involved with adaptation of components, and some of them extended the research on the field of Web service composition adaptation. It is classified as restrictive methods and generative methods according the adaptation methods [6] [8]. The former one encourages deleting the path that causes incompatibility from components behavior protocol when composing. Automatically generating adaptors without the intervention of human is one of the significant advantages of restrictive methods. However, deleting path will result in the loss of some functions of components. Generative methods mediate the incompatible behaviors through saving, reordering events and data without modifying components' behaviors. But they usually require the intervention of manual work.

The literature [6] indicated that adaptation can be classified as five layers: Technical, signature, behavioral, QoS and semantic layers. Generative methods are the mainstream in present research works. The signature and protocol layers are mainly concerned when adaptation. Based on abstract model, they can be classified as method based on finite state machine [9]-[11], method based on process algebra [12] [13], method based on Petri net [14]-[16] and method based on LTS [6] [17].

The literature [10] is the basis of present methods. Based on the description of component behavior by finite-state automata, the paper gives the definition of the behavior compatibility between two components, puts forward adaptable method for incompatible components, and discusses about how to generate adaptor depending on the analysis of all of the execution trace of finite-state automata on the basis of interfaces mapping. The literature [9] also uses finite-state automata to model Web service behavioral and generates adaptor model for duality interaction Web service according to interaction path. Because the action mapping relationship is not defined, the naming mismatch problem cannot be solved and the adaptor specification is generated with uncertainty. The literatures [12] [13] describe the component interface behavior protocol by π calculus, and then it gives out the

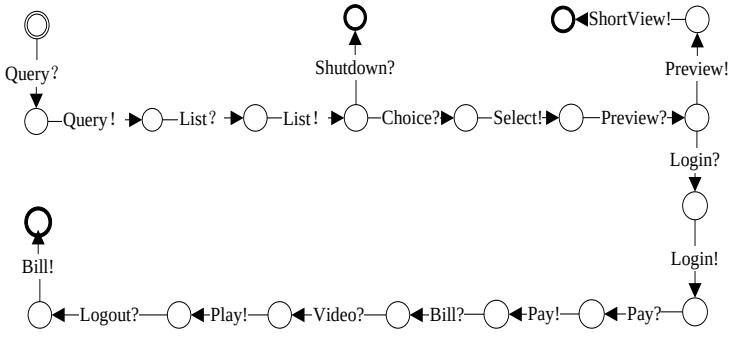

Figure 4. The LTS protocol of the VoD system adaptor.

method of generating adaptor. But the method merely works for solving duality interaction problems and the calculation complexity of is higher. The literature [14] puts forward modeling the semantic and behavior using Petri Net based on normalized structure of semantic service flow nets then converts Semantic Service Flow Nets to π calculus and provide solution for generating behavioral adaptor for Web services composition. The literatures [15] [16] use colored Petri net to model Web service behavioral and construct the reachable graph to analyze adaptable thereafter generate adaptor based on interface mapping rules. However, adaptor generated by this method neither has control logic nor to resolves adaptation under complicated scenarios. The literature [6] [17] introduce using LTS to describe components behavioral and establish adaptation contract by means of analyzing the corresponding relation between synchronous messages. It solves two mismatch situations including naming mismatch and unexpected receive. The other mismatch situations are still not being solved yet. As for the reorder mismatch problem, they model components behavior using Petri Net and establish adaptation specification. In regard of this method to generate adaptor based on interface mapping rule, one of the deficiencies is that it cannot be used to solve complicated adaptation problems. And also it requires multiple switches between LTS model and Petri Net model in confront of several mismatch situations. The document [18] suggests simulating Web service behavior through YAWL and then synthesize adaptor by analyzing the execution trace of YAWL. Specifically, transform the Web service behavioral to YAWL firstly. Then synthesize expected adaptor according to the analysis of execution trace of Web service. But both the execution trace analysis and the following adaptor verification depends on the state reachable graph of YAWL. The literature [19] defines several adaptation templates based on BPEL code. Each template corresponds to an incompatible input. The integrator generates BPEL adaptation code using adaptation template according to different Web service incompatibility situations. However, it didn't indicate how to combine these adaptation templates when confronting complicated adaptation situations.

According to the above analysis, the methods based on finite state automata are relatively easy. But they are not very well to describe complicated components or service behavior. The methods based on process algebra are relatively mature, which can be used to describe the complicated components or service behaviors even in changing topology structure. However, their complexities restrict the application of these methods in this field. Petri Net performs well in graph and expression, so it is used widely to describe various sophisticated behavior in this field. But this method depends too much on state reachable graph in generating and verifying adaptor. When confronting with large-scaled adaptation problems, it will easily causes state space explosion problem.

6. Concluding Remarks

When developing software system by compositing components, because components are provided by different third vendors, deadlock or error behavior in composited system caused by the incompatibility of semantic and behavioral layers are inevitable even if their functions meet the needs of system. Therefore, how to ensure the correct interactions among incompatible components is one of the most important tasks in CBSD. This paper models software component behavioral as LTS at first referring to literature [6]. Then stimulates the receiving and sending messages relation between interacted components by using synchronous product and judges the system compatibility by inspecting whether there is deadlock in synchronous product. When their incompatible, forming synchronous vector by analyzing their behavioral relations and generate adaptation contract according to the sequential relationship of synchronous vectors. In the following, after deleting distinct inadaptable path (s) from adaptation contract, combing the relationship of messages sending/receiving, generates adaptor specification automatically with the guide of adaptation contract so to ensure components interact correctly. The contributions of this paper are: (1) this method not only can eliminate instinct incompatible path but also generate adaptor to adaptable scenarios. It combines the advantages of restrictive and generative methods. (2) Resolves the disorder mismatch messages reordered by means of adaptor cache mechanism thus using a unified LTS model to work out different mismatch types and decrease the complexity of adaptation.

Web service composition is the extension of software component composition on Internet. The method put forward in this paper can also use to solve adaptation problems in Web service composition. How to transform Web service description to abstract model is mainly concerned when conducting Web service adaptation. That is: On the one hand, how to extract syntax layer information from WSDL automatically; On the other hand, how to generate behavioral protocol model expressed by LTS from BPEL or WF automatically.

In this paper we exploit the behavioral adaption under the condition of same operation granularity, namely the

send and receive operation relationship is one to one. For the one to many or many to one relationship is further work of research. Moreover, this paper only considers the control flow information of component behavior without concerning data flow information and how to treats them together is another research work.

Acknowledgements

This work has been funded by National Natural Science Foundation of China (NSFC) project of No. 61262012, and project No. 2008GZS0017 funded by Natural Science Foundation of Jiangxi province, China.

References

[1] Yang, F.Q., Mei, H. and Lv, J. (2002) Some Discussion on the Development of Software Technology. *Journal of Electronic*, **30**, 1901-1906.

[2] Lau, K.K. and Wang, Z. (2007) Software Component Models. *IEEE Transactions on Software Engineering*, **33**, 709-724. http://dx.doi.org/10.1109/TSE.2007.70726

[3] Plasil, F. and Visnovsky, S. (2002) Behavior Protocols for software components. *IEEE Transactions on Software Engineering*, **28**, 1056-1076. http://dx.doi.org/10.1109/TSE.2002.1049404

[4] Schlegel, C. (2007) Communication Patterns as Key Towards Component Interoperability. *Springer Tracts in Advanced Robotics*, **30**, 241-248. http://dx.doi.org/10.1007/978-3-540-68951-5_11

[5] Han, J. (2000) Temporal Logic Based Specification of Component Interaction Protocols. *Proceedings of the ECOOP'*2000 *Workshop on Object Interoperability* (*WOI*'00), June.

[6] Canal, C., *et al*. (2008) Model-Based Adaptation of Behavioral Mismatching Components. *IEEE Transactions on Software Engineering*, **34**, 546-563. http://dx.doi.org/10.1109/TSE.2008.31

[7] Deng, S.G., Li, Y., Wu, J., Kuang, L. and Wu, Z.H. (2007) Determination and Computation of Behavioral Compatibility for Web Services. *Journal of Software*, **18**, 3001-3014.

[8] Mateescu, R., *et al*. (2012) Adaptation of Service Protocols Using Process Algebra and On-the-Fly Reduction Techniques. *IEEE Transactions on Software Engineering*, **38**, 755-777. http://dx.doi.org/10.1109/TSE.2011.62

[9] Fan, D.J., Huang, Z.Q., Cao, Z.N. and Wang, J. (2012) Analysis of Behavioral Compatibility and Adaptability of Web Services. *Journal of Applied Sciences-Electronics and information Engineering*, **30**, 661-668.

[10] Yellin, D.M. and Strom, R.E. (1997) Protocol Specifications and Component Adapters. *ACM Transactions on Programming Languages and Systems*, **19**, 292-333. http://dx.doi.org/10.1145/244795.244801

[11] Chu, D.H., Meng, F.C., Zhang, D.C. and Xu, X.F. (2011) Multi-Level Component Behavior Matching Model Based on Finite Automata. *Journal of Software*, **22**, 2668-2683. http://dx.doi.org/10.3724/SP.J.1001.2011.03926

[12] Bracciali, A., Brogi, A. and Canal, C. (2005) A Formal Approach to Component Adaptation. *Journal of System and Software*, **74**, 45-54. http://dx.doi.org/10.1016/j.jss.2003.05.007

[13] Mateescu, R., Poizat, P. and Salaun, G. (2007) Behavioral Adaptation of Component Compositions Based on Process Algebra Encoding. 2007 *IEEE/ACM International Conference on Automated Software Engineering* (*ASE*), 385-388. http://dx.doi.org/10.1145/1321631.1321690

[14] Cao, G.R., Tan, Q.P. and Wu, H. (2013) An Approach to Web Service Adaptation Based on Semantic Service Flow Nets. *Journal of Chinese Computer Systems*, **34**, 2696-2701.

[15] Tan, W., Fan, Y.S. and Zhou, M.C. (2009) A Petri Net-Based Method for Compatibility Analysis and Composition of Web Services in Business Process Execution Language. *IEEE Transactions on Automation Science and Engineering*, **6**, 94-106. http://dx.doi.org/10.1109/TASE.2008.916747

[16] Li, X.T., Fan, Y.S. and Madnick, S. (2010) A Pattern-Based Approach to Protocol Mediation for Web Services Composition. *Information and Software Technology*, **52**, 304-323. http://dx.doi.org/10.1016/j.infsof.2009.11.002

[17] Poizat, P. and Salaun, G. (2007) Adaptation of Open Component-Based Systems. In *Proc. of FMOODS*'07, Springer, 141-156. http://dx.doi.org/10.1007/978-3-540-72952-5_9

[18] Brogi, A. and Popescu, R. (2007) Service Adaptation through Trace Inspection. *International Journal of Business Process Integration and Management*, **2**, 9-16. http://dx.doi.org/10.1504/IJBPIM.2007.014100

[19] Benatallah, B. and Casati, F. (2005) Grigori D. Developing Adapters for Web Services Integration. *International Conference on Advanced Information Systems Engineering* (*CAISE*), **3520**, 415-429. http://dx.doi.org/10.1007/11431855_29