Testing Conformance of BPEL Business Process Based on Model Checking

Rongsheng Dong, Zhao Wei, Xiangyu Luo, Fang Liu
School of Computer and Control, Guilin University of Electronic Technology, Guilin, China
Email:ccrsdong@guet.edu.cn

Abstract—**Formalized analysis method is a technology that insures quality of software reliability. It can detect mistakes and flaws effectively in software design. Based on the research of model checking techniques for composition of web services, we establish an automatic test framework for web services composition of BPEL. Static test method is used and test cases are generated automatically in this framework. According to the input, outputs of cases and the requirement properties, we can test the conformance for business flow of BPEL. We analyze "Airline Tickets Reservation System" which is described by BPEL with our test framework and test the conformance of the system. In addition, we compare our method to model checking method which is used to verify web services composition from three areas including states stored, searching depth and consuming times. We can conclude that our method have a better efficiency when checking the web services which owns more states.**

Index Terms—BPEL; Web services composition; model checking; testing

I. INTRODUCTION

BPEL [1] is a business process description language which can realize a complex web services composition. It defines syntax rules for the control flow and data flow of web services composition. The business flow of web services composition is designed mainly rely on the experience of designers. Since the absence of modeling and validation process, writing correct process descriptions in BPEL is not an easy task. It is very easy to cause some errors. Such as deadlock, unreachability of business functions. It is necessary to verify correctness of web services composition before implement. Testing web services composition is an effective method which can find some errors in business flow and guarantee the reasonable implement of web services composition. In this area, the main research results include the following two aspects:

A. *Testing a individual web service*

Suet Chun [2] and Jeff Offutt [3] describe the application of mutation analysis and data perturbation in the testing of web services. Their targets are the individual web services and not their composition.

B. *Testing web services composition*

Antonia Bertolino [4] propose a framework for dynamic testing of web services interoperability. They introduce a testing stage called "audition" before the services are published on a UDDI registry. In combination with verification techniques, Huang et al [5, 6] describe a method to test composite web services. They explicitly specify the web services behavior using OWL-S and define the desired properties by hand. Then, they use model checking to ascertain whether the properties hold. An efficient algorithm is presented by Zeng Yun-feng [7] to generate BPEL unit test case. Firstly, BPEL process source code is translated into BPEL flow diagram (FGBPEL) with the transformation rules, and then CTP algorithmis presented to generate test cases.

How to test web services composition with static method and less manual operation and How to generate test cases automatically? These are very concerned problems for researcher.

For these problems, according to the feature of automatically generating counter-examples in model checking from the research of Angelo Gargantini [8] and based on the research of Hai Huang [5] that web services composition of OWL-S is tested with model checking technique, an automatic testing framework is established for BPEL composition of web service based on the formalized analysis model proposed by this research study. This framework can test conformance for BPEL business flow depending on the test case of input, output and requirement properties. Ticket reservation system is taken as an example in the use of this framework to test the system's conformance. In addition, we compare our method to model checking method which is used to verify Web services composition from three areas which include states stored, searching depth and consuming times. We can conclude that our methods have a better efficiency when checking the Web services which owns more states.

Our research results include test method of based on model checking is used to test conformance of web services composition and an automatic testing framework is established for BPEL of web service composition. This framework include formalized analysis model for BPEL and provide a method transferring analysis model to Promela code. This paper is organized as follow: Section 2 introduces BPEL; Section 3 gives an automatic testing framework for BPEL of web service composition. Section 4 takes Ticket reservation system for example, the system's conformance is tested in the use of this framework and experiment results are provided. Section 5 presents a comparing for the method of test and model checking for web services composition; Section 6 presents a conclusion and discussion of future work.

II. BPEL

BPEL is a business process language based on XML. It extends WSDL and expresses the action of web services composition. It is composed of control flow and data flow. This also forms its two characteristics: behavior specification and message type. BPEL describes composition of web services with WSDL. A BPEL links with many partners described by WSDL. SOAP defines the message type. In BPEL, there are many control types, such as sequence, while, switch, etc. and many atomic work such as invoke, receive, reply (sending and receiving messages), and assign (updating the value of variable).

III. An automatic test framework for BPEL of web services composition

Using model checking tools to test Web services composition is a new method. The tool SPIN[9] of analyzing and verifying system is used in the field of software testing to test the conformance of web services composition.

Conformance is usually defined as testing to see if an implementation faithfully meets the requirements of a standard or specification. So it can check whether the implemented system realize the given system faithfully. An automated test framework is shown in Figure 1 for BPEL web services composition. Now, five steps of the framework are introduced as follow.

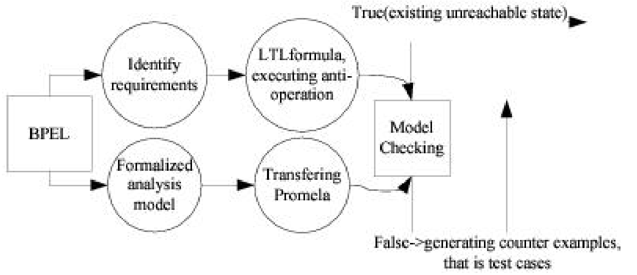
True(existing unreachable state)
Identify requirements
LTLformula, executing anti-operation
BPEL
Model Checking
Formalized analysis model
Transfering Promela
False->generating counter examples, that is test cases

Figure 1. An automated test framework for BPEL of web services composition

A. *A formalized analysis model for BPEL of web services composition*

Based on finite state automata theory [10], we define the formalized analysis model of the business flow described by BPEL. The protocol P denotes the business flow described in BPEL. An agent A denotes a partner who has two attributes: partnerLink and partnerRole. The Definitions of the formalization are as follows:

Definition 1 (*Protocol*). A protocol is the message exchange sequences between two or more agents. To define formally: A protocol $P = \langle A, M, C, \Omega \rangle$, where

- A is the set of agents,
- M is the set of messages,
- C is the communication chain,
- Ω is the set of message exchange sequences.

In Ω, each element is denoted by $r_i \xrightarrow[a_s, m_{k-1}]{m_k} r_j$, it means that when the agent r_i sends the message m_k to r_j, m_k is generated after the execution of action a_s and the prior exchange of m_{k-1}.

Definition 2 (*Agent*). Agent A is denoted by a finite state automata. It consists of six elements. To define formally: $\forall a \in A, a = \langle S, L, \delta, \Theta_{r0}, F \rangle$, where

- S is a finite set of states. A finite state automata must be in one determinate state anytime.
- L is a transitional label.
- δ is a transition relation: $\delta \subseteq S \times L \times S$, transition $\tau = (s, l, s') \in \delta(s \xrightarrow{l} s')$ means that the process executes the transition when the transition condition is true, changing from state $s \subset S$ to state $s' \subset S$. If π denotes current state in FSM ,such as $\pi = S_0$ denotes initial state; c denotes condition , if $\pi = s \,\&\&\, c == true$, a transition τ is enabled. The set of enabled transitions in a state s is denoted by $enabled(s)$. Note that when it exits empty input event, FSM will not execute any transition, and not generate any output.
- Θ_{r0} is the set of initial states for agent A . FSM can receive input from this state.
- F is the set of final states. FSM can not receive any input, when it reaches the final state.

There are at least two agents or more to exchange message in the way of synchronous or asynchronous in the chain. If it is synchronous exchange, the chain $C = \varnothing$; If it is asynchronous exchange , the change of chain C $\rightarrow^m$ is shown as follow :

When agent $\forall a_i \in A$ sends the first message in the protocol, $\rightarrow^m$ denotes as follow:

$$(\pi_i = s_i) \wedge (\exists \tau : s_i \rightarrow^l s_i' \in \delta_i / \tau \in enabled(s_i))$$
$$\Rightarrow \varnothing \rightarrow^m C^1$$

Where $C^1 = \{?m \mid m \in M_i\}$, $?m$ denotes chain C receive a message m.

When agent $\forall a_i^{;} \in A \wedge a_i^{;} \neq a_i$, and receive a message, $\rightarrow^m$ denotes as follow:

$$(\pi_i = s_i) \wedge (\exists \tau : s_i \rightarrow^l s_i^{;} \in \delta_i / \tau \in enabled(s_i))$$
$$\Rightarrow C^k \rightarrow^m C^{k-1}$$

Only when agent $a_i^{;}$ receive a message from a_i , two agent complete a message exchange sequence .

When the protocol enter the final state, $\rightarrow^m$ denotes as follow:

$$(\pi_i = s_i) \wedge (\exists \tau : s_i \rightarrow^l s_i^{;} \in \delta_i / \tau \notin enabled(s_i))$$
$$\Rightarrow C^1 \rightarrow^m \varnothing$$

A business flow of BPEL is a sequence of message “M”, which at least two agents “A” exchange message in the chain “C”. We define formalized analysis model for business flow of BPEL with P and A, It is shown as follow:

In $P = \langle A, M, C, \Omega \rangle$, agent A is a partner in BPEL ,it is signed by name, which is extracted from variable <partners> . Take Airline Tickets Reservation system[11] for example, “< partners >< partner name = "Traveler" ... / >< partner name = "TravelAgent"... / >< partner name = "Airline" ... / >< / partners >”,
So $A = \{\text{Traveler}, \text{TravelAgent}, \text{Airline}\}$.

Message M is also extracted from “<variables>”. For example, message “reserveickets” is extracted from

“< variables >
< variable name = "ordertrip" ... / >
< variable name = "checkseats" ... / >
< variable name = "reserveickets" ... / > [...]
</variables >” .In WSDL document, the data structure of message “reserveickets” is shown as follow:
“<message name=" reserveicketsM">
<part name=" ordertripID " type="xsd:int"/>
<part name=" airlineID " type="xsd:int"/>
<part name=" airline_flag " type="xsd:bool"/>
<part name=" money_ID " type="xsd:int"/>
</message>”,

So $M = \{m_0, m_1, m_2 ......\}$, In the Airline Tickets Reservation system , m_0 denotes message “ordertrip” , m_1 denotes message “checkseats”, m_2 denotes message “reserveickets”, et al.

C denotes communication chain which is extracted from “portType”; Ω is the set of message exchange sequences; $\rightarrow^m$ denotes transition of business flow of BPEL.

$\forall a \in A, a = \langle S, L, \delta, \Theta_{r0}, F \rangle$, S is a set of states in business flow of BPEL for an agent; Θ_{r0} is a set of initial states; F is a set of final states; L is a transitional label in business flow of BPEL for an agent. State transition function δ denotes activity in business flow of BPEL for an agent. For example, <receive operation=”approve” variable=”accept”/>, it means if accept=”approve_in", state will change from t1 to t2.

There are two types of activities: primitive activity and structured activity in BPEL. Now ,we represent these activities with FSM model .In this model, a transition connects two states and is labeled following the syntax “S-R-OP” , where S denotes the address of sender, R denotes port on which message to be received, OP denotes operation to be performed .

1) Primitive activities

Primitive activities include message exchange activities “<invoke>, <receive>, <reply>”, data manipulation activities “<assign>”, and others activities “<terminate>,<wait>,<empty> , <throw>”.

Message exchange activities correspond to the WSDL operations. There are four types of WSDL operations:

A: One-way operation: only receives a message without sending any response.

B: Request-response operation: receive a message and send a response back to the sender.

C: Solicit-response operation: send a message to a service and wait for a response.

D: Notification operation: send a message to another service.

Figure 2 shows the FSM models of these four types of WSDL operations:

<Invoke> Activity:

Invoking an operation can be a synchronous request/response or an asynchronous one-way operation. A synchronous invocation requires both an input variable and an output variable. An asynchronous invocation requires only the input variable of the operation.

<Receive> Activity:

A receive activity specifies the partner link it expects to receive from, and the port type and operation that it expects the partner to invoke. In addition, it may specify a variable used to receive the message data being expected.

<Reply> Activity:

A reply activity is used to send a response to request previously accepted through a receive activity. Such responses are only meaningful for synchronous interactions. An asynchronous response is always sent by invoking the corresponding one-way operation on the partner link. A reply activity may specify a variable that contains the message data to be sent in reply.

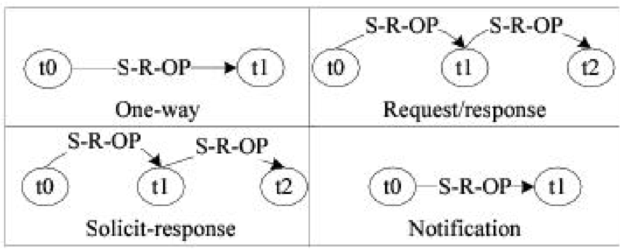

Figure 2. FSM models of four types of WSDL operations

<Assign> Activity:

The assign construct can be used to update the values of containers with new data.

<Terminate> Activity:

The terminate construct allows you to immediately terminate a business process.

<Empty> Activity:

The empty construct allows you to insert a "no-op" instruction into a business process.

<Throw> Activity:

The throw construct generates a fault from inside the business process.

Figure 3 shows the FSM models of these primitive activities:

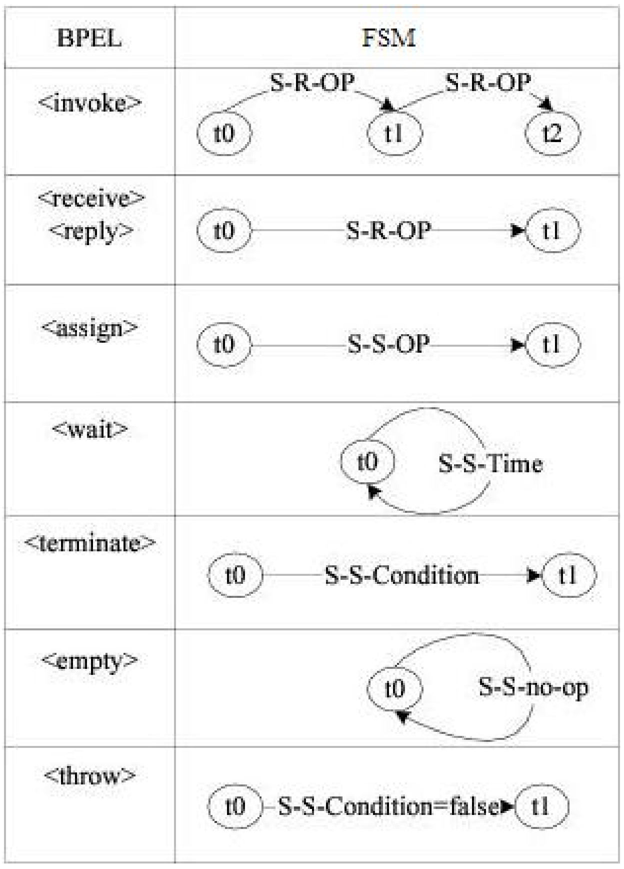

Figure 3. FSM models of primitive activities

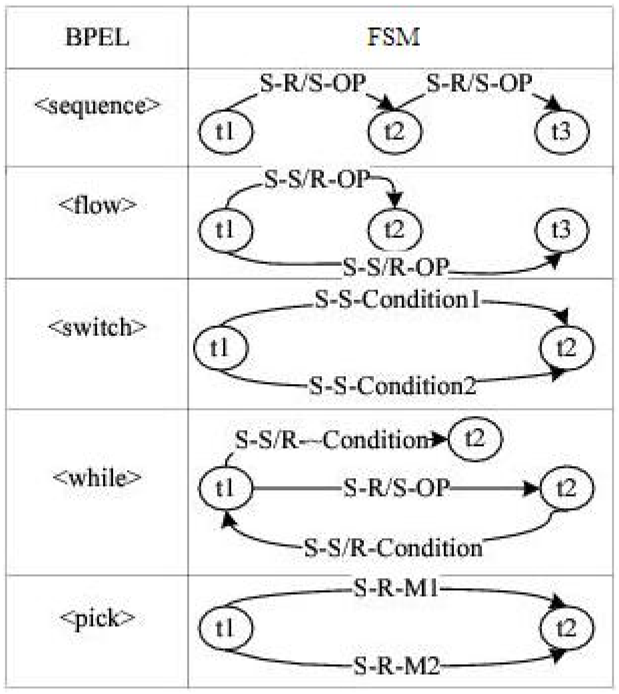

Figure 4. FSM models of structured activities

2) Structured activities

<Sequence> Activity:

Sequential execution of activities. The activities are performed in the order in which they are listed within the <sequence> element.

<Switch> Activity:

Test and branch conditions. The activity consists of an ordered list of one or more conditional branches.

<While> Activity:

Process iterations. This activity provides a construct to perform iterative execution of activities until a Boolean condition is evaluated to true.

<Pick> Activity:

Event driven selection of transitions. This activity awaits the occurrence of one of a set of events and then performs the activity associated with the event that occurred.

<Flow> Activity:

Concurrent message transitions. The flow completes when each activity contained within the flow scope has completed, and each activity is executed concurrently.

<Links> Activity:

Transitional conditions between constructs. This activity is used to determine when activity transitions can be made given the requirement that other activities have successfully completed.

<Scope> Activity: Scope sub-process of activities for compensation.

<Compensate> Activity: Force compensation.

<FaultHandlers> Activity: Define either global or scope fault handling.

Figure 4 shows the FSM models of these structured activities.

B. Translating the formalized analysis model into Promela code

Now, we translate analysis model "$P = \langle A, M, C, \Omega \rangle$, $\forall a \in A, a = \langle S, L, \delta, \Theta_{r0}, F \rangle$ " to a Promela programs. It is shown as follow:

(1)Each agent "A" is defined as a global constant, such as: #define $agent_i$ 100 .

(2)The type of corresponding messages "*M*" is defined as mtype in Promela (the definition of control flow in BPEL). mtype={xmsg1,xmsg2,…}. The content of corresponding messages (the definition of data flow in BPEL) can be defined with the following structure:

$$\begin{aligned}&typedef\ \ msg_i\\&\{bool\ data1;\\&\ \text{int}\ \ data2;\\&\ \text{int}\ \ data3;......\}\end{aligned}$$

(3) "*C*" is transformed into Promela message hannel as follow: All agents exchange message through the communication channels. Communication channels can be defined as two models. If the delay of communication is not considered and the message delivery is instantaneous, the channel can be defined as chan channel=[0] of {mtype, bool, int, …} (mtype is the messages type and "bool, int…" are the data types of the message content). Two interactive agents share one channel, one sending messages and the other receiving messages through the corresponding communication channel. If the message delivery is not instantaneous, the channel can be defined as chan channel=[N] of {mtype, bool, int, …} ($N \geq 1$) and an additional agent—message broker agent "msgagent". In order to manage all messages as a whole, every message from $agent_i$ will be firstly sent to msgagent, and then msgagent transmits every message to the corresponding agent according to the receiving object of the message in this model. The format of message sending is: $channel!agent_i, xmsg_i, msg_i$. Notice that in this paper we adopt two model of channel definition because many of the web services compositions involve synchronous and asynchronous message sending etc.

(4) Ω is a sequence of Promela programs .

(5) $\forall a \in A, a = \langle S, L, \delta, \Theta_{r0}, F \rangle$, Each agent is a FSM . It is expressed in a proctype type in Promela. An instance of a proctype is a process where Promela processes run concurrently and are executed non-deterministically.

(6)S is a set of finite states for a agent .

(7)Transitional label L denotes input and output behavior .such as, input behavior $C?X$,it means a agent receive messages from chan; output behavior $C!X$,it means a agent send messages to chan .

(8) The regulations for the transition of state behavior of each agent δ are provided as follows:

$$\begin{aligned}&state = \Theta_i, i \geq 1, \forall \Theta_i, \Theta_{i+1}, \Theta_{i+2} \in P,\\&(state == \Theta_i)\ \&\&(condition == true) \Rightarrow\\&atomic\{chanmsgagent!agenti, xmsg_i, xmsg_i; j = data_{i+1}; state = \Theta_{i+1}\}\\&(state == \Theta_i)\ \&\&(condition == true) \Rightarrow\\&atomic\{chanagent_i ? \text{var}, parm.....;\\&if :: \text{var} == xmsg_i \rightarrow j = data_{i+1}; state = \Theta_{i+1};\\&:: \text{var} == xmsg_{i+1} \rightarrow j = data_{i+2}; state = \Theta_{i+2};\\&fi;\}\end{aligned}$$

Remark: $\forall \Theta_i \in P$, Θ_i , $data_i$ are defined as constants. Suppose j, var, parm is a variable, "state" is a variable which denotes different states, and "condition" is a variable which determines if the condition is met or not.

(9) Θ_{r0} is the initial state for agent of Promela programs ; F is the final state for agent of Promela programs .

C. Identify testing specification and requirements

In order to produce test cases for BPEL, test requirements must be identified .As it has been said, this is commonly done by hand in the field of software testing. However, this paper provides a method to automated produce test cases. These properties defined by LTL and the counter-examples generated by model checking are used in this method. It is described as follow:

Figure 5 shows a finite state transition diagram .We assume a finite state system is described by this diagram and give the definition of this system for a deterministic finite state machine .

To define formally: $M = (Q, \Sigma, \delta, q_0, F,)$,where $Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7\}$ is a finite set of states , Σ ={c1,c2,……cm} is a set of input , $\delta := Q \times \Sigma \rightarrow Q$ is a state transition function , $q_0 \in Q$ is the initial state, $F \in Q \wedge F = q_7$ is the final state ,The variables $x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7$ denote the state of $q_0, q_1, ...q_7$ respectively in the system. The initial value is defined to "0" for all variables. If the state can be reach, then the value of the variable corresponding to state is changed to 1.

Let's consider how to test whether a state satisfy reachability. For example, testing the state q_1 is expressed as $[]!x_1$ in LTL. The meaning is: There is not a state q_1 which could satisfy reachability. According to the figure of the finite state system, it is possible to reach the q_1 state. Hence, we can obtain a false result with this LTL formula.

After executing the model checker, the model checking tool SPIN will search all states exhaustively in order to find a transition which is not satisfied the attributes. So, it can generate counter-examples and the counter-examples sequence may be {q0, q1}. As a result of the depth-first search in model checking, it will return and not search further if finding an error and generating

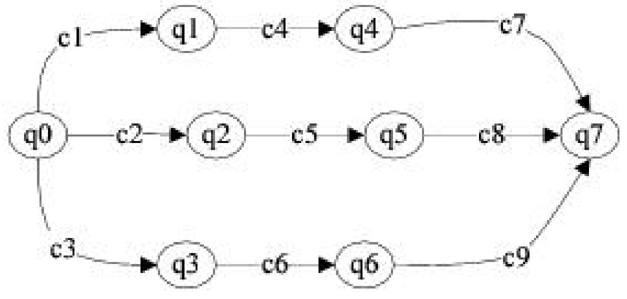

Figure 5. A finite state transition diagrams

a counter-example. So the counter-examples sequence may not include the final state.

If we need that the final state q_7 is included in counter-examples sequence, we will change the LTL formula to $[](!x_1 \| !x_7)$.The meanings is: There is not a condition q_1 and q_7 states which could satisfy reachability synchronously. In the same way, this attribute is not satisfied after execute model checker. The counter-example sequence may be $q_1 \to q_7$.

Now, we observe the change of variables $x1$ and $x7$. When the system is in the initial state, $x1 = 0; x7 = 0$, after executing model checker and generating counter-examples, $x1 = 1; x7 = 1$. In others words, the input of test cases is $x1 = 0; x7 = 0$; the output of test cases is $x1 = 1; x7 = 1$. According to the figure 5, the system may execute the state $q1$ and $q7$.So $x1 = 1; x7 = 1$. When executing model checker and generating counter-examples, $x1 = 1; x7 = 1$.It is the output of test cases. Comparing the requirements and the output of test cases, So we can decide whether the system can satisfy the conformance.

The above methods will be used in BPEL of web services composition. We conclude that the LTL formula of input and output of test cases is defined to $[](!X \| !BPEL\_End)$. (Where X denotes a state in BPEL, BPEL_End denotes the final state.) The requirement functions are obtained from user requiring book.

D. The execution of model checker

This step is an automation of the verification process. After executing the model checker, the model checking tool SPIN will search all states exhaustively in order to find a transition which is not satisfied the LTL formula. If the result is true, it means the transition is not executed. Or we can get test cases from the business flow of BPEL. In the same way, we must run SPIN many times with this method, and find all correlative counter-examples which include all transitions. In order to find all counter-examples, we must select option “Set Advanced Options-Save All Error trails” in SPIN.

E. Analysis of test cases

We analyze test cases from 3.4. According to system requirements defined in 3.3, we observe internal variables, record the initial input values and the output value generated from counter-examples. Then, we compare the system requirements and input value, output value. This is conformance test.

IV. CASE STUDIES

This paper take airline tickets reservation for example, we test the conformance of the system with the test framework. Due to the limit of space, We only present how to generate test cases, how to analyze it, and experiment results.

Test specification: For the web services of the Traveler, BOOL variable *flag_ start* is defined (its initial value is false). It denotes the executing of the state t1. In the same way, if Traveler service sends message about reserving seat, the variable value of *flag_ start* will change from false to true. *Flag_end* denotes system termination. The meanings of the Test specification is: if Traveler sends traveling message, the system will enter the termination state in the end. It is express by LTL formula as follow:

[](!(flag_start = = 1)|| !(flag_end = = 1))

Requirements: Five requiring functions are defined for airline tickets reservation system as follow:

Property P1: If a traveler sends travel information, ticket reservation system can reserve seat in the effective time, the traveler can get a ticket.

Property P2: If a traveler sends travel information, ticket reservation system can not reserve seat in the effective time, the traveler will receive a failed notification.

Property P3: If a traveler sends travel information, ticket reservation system execute the operation of canceling reserve seat, the traveler will receive a cancel notification.

Property P4: If a traveler sends travel information, there is not a available seat in ticket reservation system when the traveler send not conformed message, the traveler will receive a failed notification.

Property P5: If a traveler sends travel information, there is not a confirmed message in ticket reservation system from the traveler, the traveler will receive a timeout notification.

P1Test: This is the first test. We execute the model checker. The result is false. It means the system executed the t1 state and the final state. Because of false result, we can get a counter-example. It is a test case. We analyze the test case and transfer them to specifications including two inputs (1.The traveler decide to have a travel; 2. The airline services generate a result that there is a available seat.) and one output (Traveler receive a ticket successfully). We compare input and output from this test to requiring property P1.The result shows property P1 is satisfied. At the same way, we run SPIN many times with the same method, and find all counter-examples including all transitions.

Five test results show that five requiring properties are satisfied. So, this system is satisfied to conformance.

V. Compare Model Checking to Test Method for Web Services Composition

Airline tickets reservation system is verified with model checking by the method of literature [12] in this section. Five requiring function properties are checked. They are expressed by LTL as follow:

[]((p && r)-> <> q)

Property P1 is expressed as follow:

G((ordertrip==1&&book_seat_ok==1)
->F(receive_tickets==1))

It means that if a traveler sends travel information, and ticket reservation system can reserve seat in the effective time, the traveler will get a ticket. Executing model checker, it will not detect an error. It means this property is satisfied.

The results of test and model checking are in TABLE I (where A denotes test; B denotes model checking):

According to the data in TABLE I, we can draw the following conclusions:

For the same requiring property, comparing method of model checking and test, the performance of the former is less than the latter in the searching depth, consuming times and states stored. For property P1, It needs 20341 states to store and takes 0.671 seconds to verify with model checking. But it needs 216 states to store and takes 0.156 seconds to verify with test method.

Why? The reason is shown as follow: When we execute model checker, it will search all states exhaustively in order to find an error trail violating property. When we use the test framework to test some properties in this paper, if the result is true, it means the system exist errors of unreachability; if the result is false, it will generate a counter-example. We can check whether the system is satisfied to conformance by input, output and system requirements. Because of generating counter-examples, the model checker will terminate searching early instead search all states exhaustively. So, it reduces the verification time and the memory overhead. When checking the web services which own more states, we have a better efficiency using the test framework.

VI. Conclusion and Future Work

Based on the research of model checking techniques for composition of web services, we establish an automatic test framework for web services composition of BPEL. Static test method is used and test cases are generated automatically in this framework. According to the input, outputs of cases and the requirement properties, we can test the conformance for business flow of BPEL. We analyze "Airline Tickets Reservation System" which is described by BPEL with our test framework and test the conformance of the system. In addition, we compare our method to model checking method which is used to verify web services composition from three areas which include states stored, searching depth and consuming times. We can conclude that our methods have a better efficiency when checking the web services which owns more states. In the future work, we will establish a uniform framework which combines test and model checking method for web services composition. So, we can analyze performance of web services composition efficiently; ensure the quality of web services composition.

TABLE I.
The results of test and model checking

Property	Method	State Vector (byte)	Depth	Real time(s)	States stored
Property P1	A	740	414	0m0.156	261
	B	774	671	0m0.671	20341
Property P2	A	740	413	Om0.125	305
	B	744	671	Om0.733	20453
Property P3	A	740	413	Om0.141	395
	B	744	671	Om0.733	21125
Property P4	A	740	413	Om0.187	547
	B	744	744	Om0.733	21195
Property P5	A	740	671	Om0.249	5078
	B	744	671	Om0.951	23799

Acknowledgement

This work was partly supported by National Natural Science Foundation of China under grant No.60763004, China Postdoctoral Science Foundation under grant No.20090450389, Natural Science Foundation of Guangxi Province of China under grant No.0991242 and Young Science Foundation of Guangxi Province of China under grant No.GuiKeQing0728090.

References

[1] OASIS. Web Services Business Process Execution Language Version 2.0[EB/OL]. http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf,2007.

[2] Chun,S & J.O®utt. Generating Test Cases for XML-based Web Component Interactions Using Mutation Analysis[A].12th IEEE International Symposium on Software Reliability Engineering[C]. Hong Kong:IEEE Computer Society press.2001:200-209.

[3] Offutt,J & W.Xu. Generating Test Cases for Web Services Using Data Perturbation[N].ACM SIGSOFT Software Engineering Notes,2004(5).

[4] Bertolino,A & A.Polini, et al. The Audition Framework for Testing Web Services Inter-operability[A].31st EUROMICRO Conference on Software Engineering and Advanced Applications[C].Porto (Portugal): IEEE press.2005:134-142.

[5] Huang,H & W.Tsai, et al. Automated Model Checking and Testing for Composite Web Services[A].Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing [C].Seattle(USA):IEEE Computer Society press.2005: 300-307.

[6] Huang,H. Model Checking:Novel techniques and applications[D].Arizona:Arizona state university.2005.

[7] Zeng,Y.F & H.Zhou, et al. Research on BPEL test case generation[J].Computer Engineer and Design,2008, 29(20): 5243-5249.

[8] Gargantini,A & C.Heitmeyer. Using model checking to generate tests from requirements specications[A]. Proceedings of the 7th European Engineering Conference Held Jointly with the 7th ACM SIGSOFT International Symposium on Foundations of Software Engineering[C]. Toulouse:ACM press.1999:146-162.
[9] Holzmann, G.J. The Model checker SPIN[J].IEEE Transaction on Software Engineering,1997,(23):76-95.
[10] Hopcroft,J.E & R.Motwani,etal. Introduction to Automata Theory, Languages,and Computation(2nd Edition)[M].New York: Addison Wesley,2001.
[11] Diaz,G & J.J.Pardo. Verification of Web Services with Timed Automata[J].Electronic Notes in Theoretical Computer Science,2006,15(7):19-34.
[12] Dong,R.S & Zhao,W,Xiangyu,L. Model Checking Behavioral Specification of BPEL Web Services[A].The 2008 International Conference of Computer Science and Engineering[C].London,U.K,2008,7:198-203.
[13] QUENUM,J.G & S.AKNINE, et al. A Modelling Framework for Generic Agent Inter-action Protocols[A]. 4th International workshop on Declarative Agent Languages and Technologies N4[C]. Hakodate,Japon: Springer press.2006:207-224.
[14] Luo,X.Y & K.L.Su, et al. Bounded Model Checking for Temporal Epistemic Logic in Synchronous Multi-Agent Systems[J]. Journal of Software, 2006,17(12):2485-2498.
[15] Luo,X.Y & K.L.Su, et al. Verification of Multi-agent Systems via Bounded Model Checking[A].The 19th Australian Joint Conference on Artificial Intelligence. Volume 4304 of Lecture Notes in Computer Science[C]. Hobart:Springer press.2006:69-78.
[16] Clarke, E.M & O.Grumberg, et al. Model Checking[M].Cambridge:MIT Press,2000.

Rongsheng Dong was born Hubei, China in 1965. He is currently a professor and interested in network security, formal technology, protocol engineering, etc.

Zhao Wei was born Guilin, China in 1984. He received the BS degree in computer science and technology from Guilin University of Electronic Technology, Guilin, China in 2006 and received the ME degree in computer science and technology from Guilin University of Electronic Technology in 2009. He is currently a doctoral student in BeiHang University and interested in network security, formal technology.

Xiangyu Luo was born Guilin, China in 1974. He received the BS degree in applied mathematics from University of Electronic Science and Technology of China in 1996 and received DE degree in computer software and theory from National Sun Yat-sen University in 2006. He is currently an associate professor and interested in agent, network security, etc.

Fang Liu was born Guilin, China in 1984. She received the BS degree in computer science and technology from Guilin University of Electronic Technology, Guilin, China in 2007. He is currently a postgraduate and interested in network security, formal technology.