Testing Service Composition Using TGSE tool

Dung Cao, Patrick Felix, Richard Castanet, Ismail Berrada

Testing Service Composition Using TGSE tool*

Tien-Dung Cao [1], Patrick Félix [1], Richard Castanet [1] and Ismail Berrada [2]
[1]LaBRI - CNRS - UMR 5800, University of Bordeaux 1
351 cours de la libération, 33405 Talence cedex, France.
Email: {cao,felix,castanet}@labri.fr
[2]L3I, La Rochelle University, 17042 La Rochelle, France.
Email: ismail.berrada@univ-lr.fr

Abstract

This paper proposes an approach to test (actively and passively) a Web service composition described in BPEL using the TGSE (Test Generation, Simulation and Emulation), that is a tool to generate the test cases based on the Communicating System (CS) and implementing of a generic algorithm of generation. For the first step, the BPEL specification is transformed into Timed Extended Finite State Machines (TEFSM) model which enable modeling of BPEL behaviour, timing constraints, its data variables and clocks. On the active testing approach, test case generation is based on simulation where the exploration is guided by test purpose that is a part of CS (i.e. it is also modeled by a TEFSM). Otherwise, the passive testing approach, the TGSE tool will verify a trace that is either correct or incorrect with specification. This tool also works with the timing constraints on clocks (local and global). It covers test cases not only with transitions but also with values of data variables. Our method is illustrated by an example.

1 Introduction

BPEL (Business Process Execution Language) [1] is an emerging standard language to describe web service composition behaviour. A BPEL process implements one Web service by specifying its interactions with other Web services (called partner services). Various approaches for service composition testing were analyzed by [2] including unit testing, integration testing, black box testing and white box testing of choreographies and orchestrations. On the active testing approach, several test case generations for BPEL Web services recently have been proposed [6, 7, 9, 10, 20], even though timed test cases [16, 17, 21]. These methods have been given pertinent test cases. On the contrary, passive testing, several techniques have been proposed [24, 25]: these methods focus on monitoring techniques and diagnosis semantic fault of BPEL service. The time notion is not considered in these methods.

The TGSE tool (Test Generation, Simulation and Emulation) [22, 23] that is composed of a test case generator based on the Communicating Systems and implementing of a generic algorithm of generation. This tool is developed by LaBRI within the RNRT Avérroes project and the European project Marie Curie RTN TAROT (MCRTN 505121). This tool follows us to active testing (i.e. test generation from one or some of components that are described by TEFSM) and passive testing (i.e. verify a trace) and it also works with the timing constraints on clocks(local and global). It covers test cases not only with transitions but also with values of data variables. This is helpful for passive testing approach.

A. Bucchiarone et al. [2] have defined two approaches for Web services composition testing:

- ***White box*** approach: in this approach, as BPEL is an executable language, the BPEL description of Web services composition is considered as the source code of the composition. It is executed by any BPEL engine (Active BPEL, Oracle...). Several structural criteria of coverage based on the code can be applied;
- ***Black box*** approach: in this approach, a composite Web service is actually coded in a different language from the specification. For instance, a BPEL specification coded as a Java program. An implementation of a composite Web service is tested without any information of its internal structure. The test suite generated from only specification.

In this paper, we focus on *Black box* testing of an implementation of the Web services composition that described in BPEL. For the first step, we present a timed modeling of BPEL based on Timed Extended Finite State Machine

*This Research is supported by the French National Agency of Research within the WebMov Project http://webmov.lri.fr

(TEFSM) to automatic testing. The TEFSM formalism allows to deal not only BPEL behaviour but also timing constraints, its data variables and clocks. In this model, we assign a time invariant for each timed activity of BPEL (for example: *wait*). We give also a description how to translate BPEL specification into TEFSMs in this paper. On the active testing approach: test case generation is based on simulation where the exploration is guided by test purposes that is a part of CS (i.e. it is also modeled by a TEFSM). The TEFSM of BPEL specification and test purpose will be modeled into a communicating system that is an input format of TGSE tool. Otherwise, the passive testing approach, the TGSE tool will verify a trace that is either correct or incorrect with specification. This trace is also modeled as a TEFSM and it is a component of CS.

The remainder of this paper is organized as follows. Section 2 reviews some previous work on Web services composition testing. The section 3, we give some of definition about: TEFSM that is used to model BPEL process, a partial of TEFSM and a Communicating System. The section 4 describes the relationship between BPEL conceps and TEFSM. How to test a service composition using the TGSE tool is introduced in the section 5. A case study is studied in the section 6. The section 7 concludes the paper.

2 Related Works

In the last years, several techniques and tools have been developed to test Web services. Various approaches for service composition testing were analyzed by [2] including unit testing, integration testing, black box testing and white box testing of choreographies and orchestrations. Jose Garcia-Fanjul et al [6] use a formal verification tool, the SPIN model checker, to generate test suite specifications for compositions specified in BPEL. A transition coverage criterion is employed to define a systematic procedure to select the test cases. Yongyan Zheng and Paul Krause [7] model each BPEL activity into an automaton (also referred as Web Service Automaton). After that, these models are transformed into Promela that is input format for model checker SPIN. In the paper [10], the SPIN model checker is used one more time to verify BPEL, but the authors do not transform directly BPEL into Promela as in [6]. BPEL will be translated to guard condition which it is transformed to Promela. In all of these methods, a test case is generated from *counterexample* that is an output of SPIN model checker. Transforming BPEL into Intermediate Format Language (IF) is presented in [17]. Thus, the timed test case is generated using TestGen-IF tool. In our previous work [20, 21] some of test case generation methods were proposed for white-box testing approach. Actually, it is not supported by any toolset. Some of work presented frameworks for white box testing without model checker. [27, 28] presented a framework for white box testing of BPEL composition. The problem of [27, 28] is that they do not consider the automatic test case generation [17].

On the passive testing approach, several methods have been proposed. [24, 25] present an automatic method to model Web service behaviors and their interactions as a set of synchronized discrete-event systems. This modeling is the first step before tracing the evolution of the business process and diagnosing business process faults. This method is based on the fact that the existing Model Based Diagnosis (MBD) techniques in Artificial Intelligence provide ways to monitor and diagnose static and dynamic systems using partial observations. The timing constraints are not considered in these methods.

3 Preliminaries

We introduce in this section the formal definition of TEFSM that is used to model BPEL, the partial of TEFSM and the communicating system.

Definition 1 *(TEFSM): A machine TEFSM M with invariant is defined as a sextuple, M = (S, s_0, V, E$\cup\{\epsilon\}$, C, Inv, T) where:*

- S = $\{s_0, s_1, ..., s_n\}$*: A finite set of states;*
- $s_0 \in$ *S: initial state;*
- *V : A finite set of data variables where:* $\vec{v} = (v_0, v_1, ..., v_m)$*;*
- *E : A finite set of the events including symbols below:*
  - ***?pl.op.msg**: input event i.e the reception of the message (**msg**) for the operator (**op**) from the partner (**pl**);*
  - ***!pl.op.msg**: output event i.e the emission of the message (**msg**) for the operator (**op**) to the partner (**pl**);*
- *C : A finite set of clocks including a global clock where:* $\vec{c} = (c_0, c_1, ..., c_n)$*;*
- *Inv : S* $\mapsto \Phi$*(C) assigns a set of time invariants (logical formulas) to the states;*
- $T \subseteq S \times E \times P(\vec{v}) \wedge \phi(\vec{c}) \times 2^C \times \mu \times S$ *is a set of transitions relation where:*
  - $P(\vec{v}) \wedge \phi(\vec{c})$*: guard condition is logical formula on data variables and clocks;*
  - $\mu(\vec{v})$*: Data variable update function;*
  - 2^C*: Set of clocks to be reset;*

Example 1 *Each $t \in T$ is the form: s <e, [g], $\{c;f\}$> s' with: $e \in E$; s, s' $\in S$; $f \in \mu(\vec{v})$; $\{c\} \in 2^C$ and $g \in P(\vec{v}) \wedge \phi(\vec{c})$.*

If the machine at state *s*, an event *e* arrives and guard condition *g* satisfies. It changes to state *s'*, the clock *c* will be reseted and the function *f* is enabled to update the variables.

Definition 2 *(Partial of TEFSM): Let a TEFSM M, the partial of M is PM = (S, s_{in}, S_{out}, V, E, C, Inv, T) where: (S, s_{in}, V, E, C, Inv, T) is a TEFSM and $S_{out} \subset$ S.*

A partial of TEFSM [15] is a TEFSM extended by input state (representing the entering state of the partial machine and which replaces the initial state s_0) and a set of output states, S_{out} (representing the exit state of the partial machine).

Definition 3 *(Communicating System): A Communicating System (CS) is a 5-tuple CS=(SP, SV, R, $M_{i,1\leq i\leq n}$, TP) where:*

- *SP: A finite set of shared parameters;*
- *SV: A finite set of shared variables;*
- *R: A finite set of rules where: $\vec{r}$ is a vector n+1 elements;*
- *M_i = ($S_i, s_{0i}, V_i, E_i, C_i, Inv_i, T_i$): An automaton;*
- *TP: Test purpose;*

A Communicating System declares a set of shared resources (parameters and variables), a set of automatons and a set of rules that declare the synchronous actions between automatons, and a test purpose that is modeled as an automaton.

4 Relationship between BPEL concepts and TEFSM

BPEL [1] provides constructs to describe complex business processes that can interact synchronously or asynchronously with their partners. A BPEL process always starts with the *process* element (i.e the root of the BPEL document). It is composed of the following children: *partnerLinks, variables, activities* and the optional children: *faultHandlers, eventHandlers, correlationSets*. These children are concurrent. We will model a BPEL process as a communicating system with three automatons (activity, faultHandler, eventHandler) and we use the rules to declare synchronous actions between them. We use a *stop* variable for activities machine to terminate the rest activities if this machine happens a fault or the termination is activated by an *exit* activity. The *scope* activity will be model as a *process*.

4.1. Messages

A BPEL variable is always connected to a message from a WSDL description of partners. In BPEL, a Web service that is involved in the process is always modeled as a *porType* (i.e. abstract group of operations (noted *op*) supported by a service). These operations are executed via a *partnerlink* (noted by *pl*). In our formalism, for instance, the input message *?pl.op.v* denotes the reception of the message *op(v)* (constructed from the operation *op* and the BPEL variable *v*) via the channel *pl*.

4.2 Basic Activities

Basic activities are: *receive, reply, invoke, assign, wait, empty, exit, throw*. Each basic activity is described by a partial machine. To synchronize the faults with *faultHandler* machine, we can add two transitions *!fault* and *?done* into each partial machine if *faultHandler* activity of process exists.

The Receive Activity: <receive partnerLink=pl portType=pt operation=op variable=msg>
PM = ($\{s_{in}, s_{out}\}$, s_{in} , $\{s_{out}\}$, {*v, stop*}, {*?pl.op.msg*}, $\{c\}$, $\{(s_{in}, true), (s_{out}, true)\}$, $\{t_1\}$)

- t_1=(s_{in},<*?pl.op.msg,[stop=false],*{*c,v=msg*}>,s_{out})

The Reply Activity: <reply partnerLink=pl portType=pt operation=op variable=msg faultName=fault>
PM = ($\{s_{in}, s_{out}\}$, s_{in} , $\{s_{out}\}$, {*stop*}, {*!pl.op.msg*}, {c}, $\{(s_{in}, true), (s_{out}, true)\}$, $\{t_1, t_2\}$)

- t_1=(s_{in},<*!pl.op.msg,[stop=false],*$\{c\}$>,s_{out})
- t_2=(s_{in},<*!pl.op.fault,[ ],*{*c,stop=true*}>,s_{out})

The Assign Activity: <assign> <from> v_2 </from> <to> v_1 </to> ... </assign>
PM = ($\{s_{in}, s_{out}\}$, s_{in} , $\{s_{out}\}$, {v_1,v_2,...,v_n,*stop*}, {$\emptyset$}, {c}, $\{(s_{in}, true), (s_{out}, true)\}$, $\{t_1\}$)

- t_1=(s_{in},< _ ,*[stop=false],*{*c*, v_1=v_2,...}>,s_{out})

The Wait Activity: <wait (for=d | until=dl)>.

- <wait for=d>: PM = ($\{s_{in}, s_{out}\}$, s_{in} , $\{s_{out}\}$, {*stop*}, {$\emptyset$}, {c}, {(s_{in},c$\leq$d),($s_{out}, true$)}, $\{t_1\}$)
  - t_1=(s_{in},< _ ,*[c=d & stop=false],*$\{c\}$ >,s_{out})
- <wait until=dl>: PM = ($\{s_{in}, s_{out}\}$,s_{in} , $\{s_{out}\}$, {stop}, {$\emptyset$}, {gc}, {(s_{in},gc$\leq$dl),($s_{out}, true$)}, $\{t_1\}$)
  - t_1=(s_{in},< _ ,*[gc=dl & stop=false],*{$\emptyset$} >,s_{out})

The Throw Activity: <throw faultName=fault/>
PM = ($\{s_{in}, s_{out}\}$, s_{in}, $\{s_{out}\}$, {*stop*}, {$\emptyset$}, {$\emptyset$}, $\{(s_{in}, true), (s_{out}, true)\}$, $\{t_1\}$)

- t_1=(s_{in},<*!fault,[],*{*stop=true*}>,s_{out})

The Exit Activity: <exit/>
PM = ({s_{in}, s_{out}}, s_{in} , {s_{out}}, {*stop*}, {∅}, {∅}, {$(s_{in}, true)$, $(s_{out}, true)$}, {t_1})

- t_1=(s_{in},< *-,[],*{*stop=true*}>,s_{out})

The Invoke Activity: <invoke partnerLink=pl portType=pt operation=op inputVariable=msg_in outputVariable=msg_out>
PM = ({s_{in}, s_1, s_{out}}, s_{in} , {s_{out}}, {*v_in, v_out, stop*}, {*!pl.op.msg_in, ?pl.op.msg_out*}, {*c*}, {$(s_{in}, true)$, $(s_1, true)$ $(s_{out}, true)$}, {t_1, t_2})

- t_1=(s_{in},<*!pl.op.msg_in,[stop=false]*,{c}>,s_1)
- t_2=(s_1,<*?pl.op.msg_out,[stop=false]*,{c, *v_out=msg_out*}>,s_{out})

The Empty Activity: <empty/>
PM = ({s_{in}, s_{out}}, s_{in} , {s_{out}}, {*stop*}, {∅}, {c}, {$(s_{in}, true)$, $(s_{out}, true)$}, {t_1})

- t_1=(s_{in},< *-,[stop=false]*,{c}>,s_{out})

4.3. Structured Activities

Structural activities: structural activities are *sequence, while, switch, flow, pick, repeatUntil, if* and *scope*. They take some partial machines $PM_{i,i\in[0,n]}$ (see Fig 1) and combine them to a new partial machine.

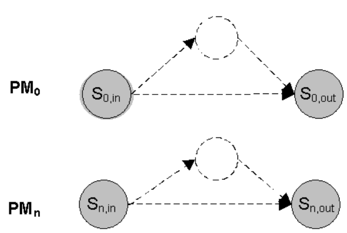

Figure 1. Partial machines

The partial machines of structural activities (*sequence, while, switch and pick*) are shown in Fig 2. The *repeatUntil* activity will be modeled as a *while* activity. The conditional behavior *if* will be also modeled as a *switch* activity. The *eventHandler* activity will be model as pick activity. The *flow* activity allows to specify one or more activities to be performed concurrently [1]. It specifies the parallel execution of the flow partial TEFSM . The *links* defined in the flow activity permit to enforce precedence between these activities, i.e. it permits synchronization. Fig 3 models links of *flow* activity.

The *faultHandlers* element combines a switch activity applied to various sequences of a *catch* or a *catchAll* activities and a sub-activities partial machine. The *catchAll*

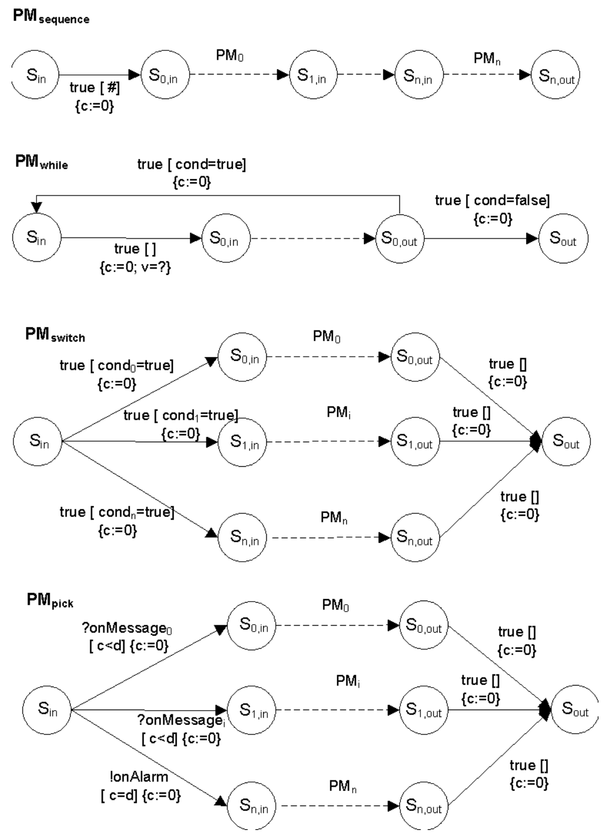

Figure 2. Modeling structural activities

element is used to catch all the faults that are not handled by the defined catch activities. Fig 4 models a *faultHandler* activity.

4.4. Limitations

There are the limitations of transformation that is described in this paper. For example: the attributes *joinCondition*, *supressJoinFailure* of the *flow* activity. An activity with *correlation* will be model by adding a variable *status* of properties as [16]. In that case, we add two transitions *!fault* and *?done* into the partial machine to handle the fault because the standard fault *correlationViolation* must be thrown [1] (i.e. synchronize the fault with *faultHandler* machine).

5 Testing Services Composition Using TGSE

In this section, we study how to test the service composition using TGSE tool. Two approaches: active testing and passive testing will be considered.

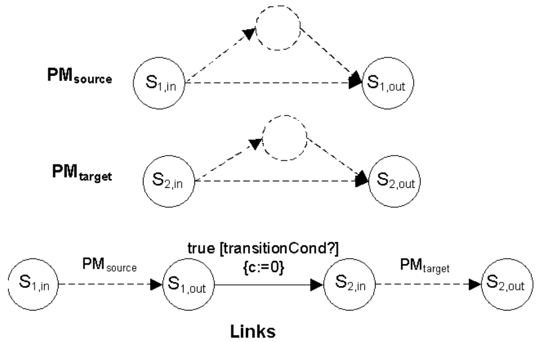

Figure 3. Modeling Links

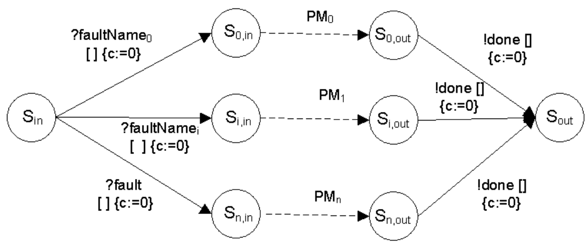

Figure 4. Modeling faultHandler

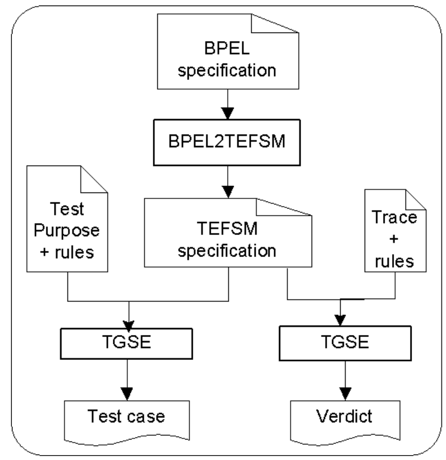

Figure 5. Overall Method of Testing Services Composition

5.1 An Overview of Testing Services Composition Using TGSE

Our method use a TGSE tool, that is a generic tool for testing, to test a service composition described in BPEL on two approaches: active testing and passive testing. The first step is to transform the BPEL description into TEFSM. Transformation can be done automatically by a prototype tool (or by hand) follow mapping rules in the section 4. On active testing approach, a test purpose is requested to generate test case. On passive testing approach, we use TGSE to verify a trace either correct or incorrect. Fig 5 illustrates the overall our methodology.

The current version of TGSE does not supported time invariant on state. Thus, we use only timing constraints on transition. Moreover, it only supported data single: integer and boolean. We use many variables to model a BPEL message.

5.2 Test Case Generation

For the purpose active testing, we use this tool to generate test cases based on test purposes. These test purposes will be modeled as a TEFSM and its actions will be synchronized with corespondent action in each TEFSM. This tool works also with timing constraints, Thus, we can use a timed test purposes (i.e. test purpose with some timed requirements) to generate timed test cases [17]. In the case, *faultHandler* activity and *compensationHandler* activity of a BPEL process exist, a communicating system for a process composes three TEFSMs ($M_{activities}, M_{faultHandler}, M_{compensationHandler}$) and a test purpose. A vector $\vec{r}$ of rules set has four elements. TGSE generates a test case in XML format file if all test purposes are satisfied, else, we have nothing. If transition condition of TEFSM depends on input value of messages, we will use a parameter as a value. We can also use a real value for the variable if test purpose requires the conditions on variable.

5.3 Passive Testing

Passive testing means that we verify a trace is either correct or incorrect with specification. Because of TGSE follows us to cover the transition based on guard condition that is examined by variable value at run time. This is very helpful for our purpose of passive testing a BPEL service. We will model a trace as a test purpose (i.e. also referred as a TEFSM) with value of each input message that is assigned by real values in the trace. In that case, all of variable of BPEL process is used as shared variables. After modeling BPEL specification and the trace as the TEFSMs and declaration its rules, we use TGSE to verify this system. If TGSE responds a sequence, it means this trace is correct with specification, else, it is incorrect.

6 A Case Study

In this section, we study an example of the Loan Web Service that is described in Fig 6. This process receives an input from the client. If this *input* is less than 10, it invokes the synchronous Assessment Service and receives a *risk* result. In the case, this *risk* is *low*, it sends a response *yes* to client. Else (*input≥10 or risk!=low*), it invokes the asynchronous Approval Service by sending a request and uses a BPEL *pick* activity for one of the following cases: (1) to receive an asynchronous response from the partner service and send this response to client; (2) to send a timeout fault to client if there is not response from the partner service after a duration (e.g., 60 seconds).

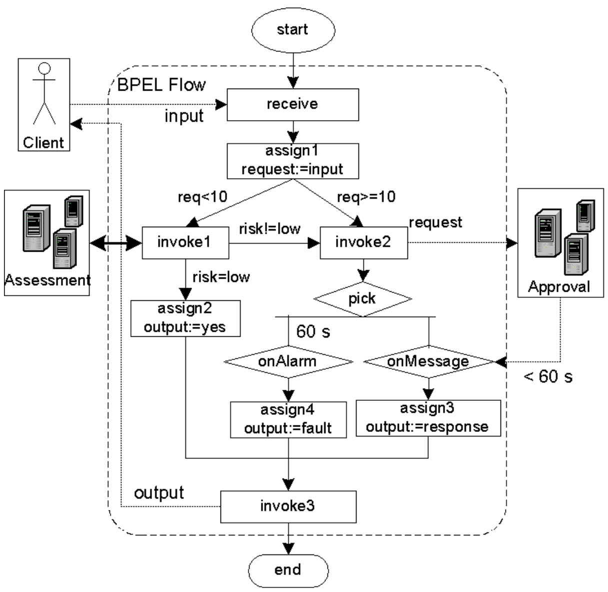

Figure 6. The Loan Web Service

6.1 The TEFSM Specification of the Loan Service

Using the rules in the section 4, we have a TEFSM of Loan Web Service in the figure 7 (The separate lines denote as transitions of link variables).

In TGSE, an TEFSM is described by: a number of state, initial state, variables of clock and a list of transition. Each transition *t* composes:

1. source_state(id, name);
2. target_state(id, name);
3. event (*nop* denotes internal event);
4. guard condition on clocks (# denotes true);
5. guard condition on variable (# denotes true);
6. reset clocks (# denotes empty);
7. update variable (# denotes empty);

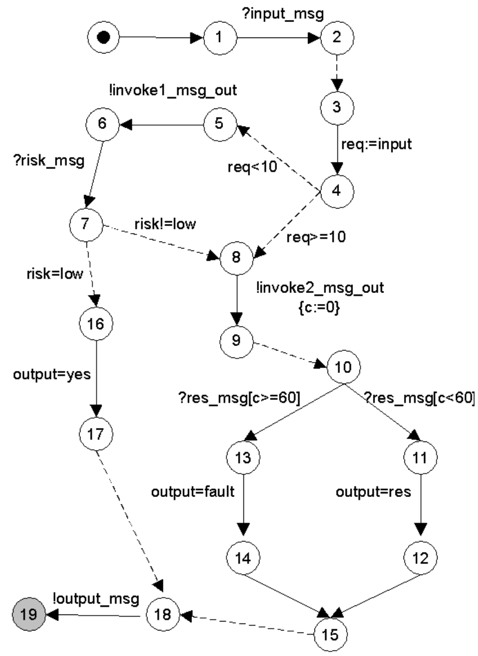

Figure 7. TEFSM of Loan Web Service

The table 1 describes TEFSM of Loan Web Service in TGSE input format. The value of variables: *request ,risk* and *response* of Approval service is used as the parameter (i.e. *p_input, p_risk and p_res*).

6.2 Test Purposes

6.2.1 Test purposes for Scenario #1

The Loan process is initiated by receiving an input from the client. It continues receive the response (i.e. risk_msg) of the Assessment service. Finally, it sends this response to the client. The test purposes for scenario #1 is formulated in TGSE as Fig 8:

TESTER test_purpose1
{
nb_states = 4
initial_state = 0
final_state = 3
(0, init), (1, state1), ?input, #, #, #, #
(1, state1), (2, state2), ?risk,#, #, #, #
(2, state2), (3, finish), !output, #, #, #, #
}

Figure 8. Test Purpose of Scenario #1 in TGSE

The rules list for this test purpose as: {<*?input_msg, ?input*>, <*?risk_msg,?risk*>, <*!output_msg, !output*>}.

Table 1. TEFSM specification of the Loan Service for TGSE

```
P_AUTO bpel
{
nb_states: 20
initial_state: 0
clocks: t

(0,init), (1,receive_in),     nop, #, #, #, #
(1,receive_in), (2,receive_out), ?input_msg, #, #, #, #
(2,receive_out), (3,assing1_in), nop, #, #, #, #
(3,assing1_in), (4,assing1_out), nop, #, #, #, req=p_input
(4,assing1_out), (5,invoke1_in), nop, #, req[-inf,10[, #, #
(5,invoke1_in), (6,invoke1_s1), !invoke1_msg_out, #, #, #, #
(6,invoke1_s1), (7,invoke1_out), ?risk_msg, #, #, #, risk=p_risk
(4,assing1_out), (8,invoke2_in), nop, #, req[10,+inf], #, #
(8,invoke2_in), (9,invoke2_out), !invoke2_msg_out, #, #, #, #
(9,invoke2_out), (10,pick_in), nop, #, #, h:=t, #
(10,pick_in), (11,assign3_in), ?res_msg, t[0,60[, #, #, #
(10,pick_in), (13,assign4_in), ?res_msg, t[60,+inf], #, #, #
(11,assign3_in), (12,assign3_out), nop , #, #, #, out=p_res
(13,assign4_in), (14,assign4_out), nop , #, #, #, out=-1
(12,assign3_out), (15,pick_out), nop, #, #, #, #
(14,assign4_out), (15,pick_out), nop, #, #, #, #
(7,invoke1_out), (8,invoke2_in), nop, # , risk[1,1] , #, #
(7,invoke1_out), (16,assing2_in), nop, # , risk[0,0] , #, #
(16,assing2_in), (17,assing2_out), nop, #, #, #, out=1
(17,assing2_out), (18,invoke3_in), nop, #, #, #, #
(15,pick_out), (18,invoke3_in), nop, #, #, #, #
(18,invoke3_in), (19,invoke3_out), !output_msg, #, #, #, #
}
```

6.2.2 Test purposes for Scenario #2

The Loan process is initiated by receiving an input from the client. It receives the response of the Approval service after 30 seconds. Finally, it sends this response to the client. The test purposes for scenario #2 is formulated in TGSE as Fig 9:

The rules list for this test purpose as: {<*?input_msg, ?input*>, <*?res_msg,?res*>, <*!output_msg, !output*>}.

6.3 Test Cases

The test cases that are generated by using TGSE is a detail path (i.e. it covers also internal actions). For instance, test case for scenario #1 as:

$$0 \xrightarrow{nop} 1 \xrightarrow{input\_msg} 2 \xrightarrow{nop} 3 \xrightarrow{nop} 4 \xrightarrow{nop} 5 \xrightarrow{invoke1\_msg\_out} 6 \xrightarrow{risk\_msg} 7 \xrightarrow{nop} 16 \xrightarrow{nop} 17 \xrightarrow{nop} 18 \xrightarrow{output\_msg} 19.$$

In our case, we focus on black-box testing, it means that we covers only the input events and the output events. We do not interested in the internal events. So that, from result of TGSE, we will cover the input events and the output

```
TESTER test_purpose2
{
   nb_states = 4
   initial_state = 0
   clocks: t
   final_state = 3
  (0, init), (1, state1), ?input, #, #, #, #
  (1, state1), (2, state2), ?res, t[30,30], #, #, #
  (2, state2), (3, finish), !output, # , #, #, #
}
```

Figure 9. Test Purpose of Scenario #2 in TGSE

events to have a test case. The test case for each scenario is shown in Fig 10.

TEST CASE 1 FOR SCENARIO #1
1. *?input_msg*
2. *!invoke1_msg_out*
3. *?risk_msg*
4. *!output_msg*

TEST CASE 2 FOR SCENARIO #2 *(first time)*
1. *?input_msg (p_req=0)*
2. *!invoke1_msg_out*
3. *?risk_msg (p_risk=1)*
4. *!invoke2_msg_out*
5. *?res_msg (p_res=0)*
6. *!output_msg*

TEST CASE 2 FOR SCENARIO #2 *(second time)*
1. *?input_msg (p_req=10)*
2. *!invoke2_msg_out*
3. *?res_msg (p_res=0)*
4. *!output_msg*

Figure 10. The Abstract Test Cases

Note 1 *TGSE has sixteen modes (i.e. from p0 to p15) that concern to select the order of transitions, automatons, and rules in the specification file. Here, we used mode p15 (all random) to run this example. The value of parameters that is generated randomly saved in the OutputLp.out file.*

7 Conclusions

We have presented a methodology using the TGSE tool to test a Web Service Composition that is described in BPEL language. Two test approaches are considered: active testing (generate the test cases) and passive testing (verify a trace). We given some of definitions on Timed Extended Finite State Machine (TEFSM), a partial of TEFSM and a Communicating System (CS). TEFSM that we proposed can enable modeling of BPEL behaviour, timing constraints, its data variables and clocks. We also define some of rules to transform a BPEL specification into TEFSMs that is the components of a CS. The Loan Web Service example is used to illustrate for our method. Our future works is to use this tool to handle integration testing as will as the choreography of Web services.

References

[1] OASIS. Web Services Business Process Execution Language (BPEL) Version 2.0, April 2007. http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html

[2] A. Bucchiarone, H. Melgratti, and F. Severoni, *"Testing Service Composition"*, In Proceedings of ASSE07,Mar del Plata, Argentina, Aug 2007.

[3] A. Gill, "Introduction to the Theory of Finite-State Machines", *Published by McGraw-Hill Book Co..*, New York, 1962.

[4] R. Alur, D. L. Dill, "A Theory of Timed Automata", *Theory of Computer Science* .vol 126, no 2, pp 183 - 235 , 1994.

[5] R. Alur, D. L. Dill, "Automata-theoretic Verification of Real-Time Systems", *In Formal Methods for Real-Time Computing*. Nov 1995.

[6] Jose Garcia-Fanjul, Javier Tuya, Claudio de la Riva, "Generating Test Cases Specifications for BPEL Compositions of Web Services Using SPIN", *International Workshop on Web Services Modeling and Testing*. 2006.

[7] Y. Zheng, P. Krause, "Automata Semantics and Analysis of BPEL", *International Conference on Digital Ecosystems and technologies*. DEST 2007.

[8] Y. Zheng, J. Zhou, P. Krause, "Analysis of BPEL Data Dependencies", *EUROMICRO Conference on Software Engineering and Advanced Applications*. SEAA 2007.

[9] Y. Zheng, J. Zhou, P. Krause, "A Model Checking based Test Case Generation Framework for Web Services", *International Conference on Information Technology*. ITNG 2007.

[10] X. Fu T. Bultan J. Su, "Analysis of Interacting BPEL Web Services", *International Conference on World Wide Web*. May 17 - 22, 2004, New York, USA.

[11] A. Wombacher, P. Fankhauser, and E. Neuhold, "Transforming bpel into annotated deterministic Finite state automata for service discovery" *Procs of ICWS04*, 2004.

[12] R. Kazhamiakin, P. Pandya, and M. Pistore, "Timed modeling and analysis in web service compositions", *The First International Conference on Availability, Reliability and Security*, vol. Volume 0, pp. 840 846, 2006.

[13] E. Bayse, A. Cavalli, M. Nunez, F. Zaidi, "A Passive Testing Approach based on Invariants: Application to the WAP", *Computer Networks*, 48, pp 247 - 266, 2005.

[14] A. Cavalli, Edgardo Montes De Oca, W. Mallouli, M. Lallali, "Two Complementary Tools for the Formal Testing of Distributed Systems with Time Constraints", *12th IEEE International Symposium on Distributed Simulation and Real Time Applications*, Canada, Oct 27-29, 2008.

[15] M. Lallali, F. Zaidi, A. Cavalli, "Timed modeling of web services composition for automatic testing", *3rd ACM/IEEE International conference on Signal-Image technologies and Internet-Based Systems*, China, 16-19 Dec 2007.

[16] M. Lallali, F. Zaidi, A. Cavalli, "Transforming BPEL into Intermediate Format Language for Web Services Composition Testing", *The 4th IEEE International Conference on Next Generation Web Services Practices*, 2008.

[17] M. Lallali, F. Zaidi, A. Cavalli, Iksoon Hwang, "Automatic Timed Test Case Generation for Web Services Composition", *Sixth European Conference on Web Services*. Dublin, Ireland, Nov 12 - 14, 2008.

[18] T. Higashino, A. Nakata, K. Taniguchi and A. Cavalli, "Generating Test Case for a Timed I/O automaton model", *International Workshop on Testing of Communicating Systems*. Budapest, Sep 1999.

[19] A. Hessel et P. Pettersson, "A Global Algorithm for Model-Based Test Suite Generation", *3rd Workshop on Model-Based Testing*, Braga, Portugal, Mar 31 - Apr 1, 2007.

[20] T. D. Cao, P. Felix, R. Castanet, "Generating Test Cases for BPEL Web Services Based on Extended Finite State Machine Model", *The 1st International Workshop on Advanced Techniques for Web Services (AT4WS 2009)*.Milan, Italy, 6 - 7 May, 2009, Submitted.

[21] T. D. Cao, P. Felix, R. Castanet, "Generating Timed Test Case Suites for BPEL Web Services", *1st IEEE Workshop on Performance Evaluation of Communications in Distributed Systems and Web based Service Architectures*. Sousse, Tunisia, 5 - 8 July, 2009, Submitted.

[22] I. Berrada and P. Félix, "TGSE : Un outil générique pour le test", *Proc. of CFIP'2005*, March, 2005.

[23] I. Berrada, "Modélisation, Analyse et Test des systems communicants contraintes temporelles : Vers une approche ouverte du test", *PhD thesis of University Bordeaux 1*, December 2005.

[24] Y. Yan, Y. Pencole, M.O. Cordier, A. Grastien, "Monitoring Web Service Networks in a Model-based Approach", *3rd European Conference on Web Services (ECOWS)*, Vxj, Sweden. 14 - 16 Nov 2005.

[25] Y. Yan, P. Dague, "Monitoring and Diagnosing Orchestrated Web Service Processes", *Proceedings of the 2007 IEEE International Conference on Web Services (ICWS 2007),* Salt Lake City, Utah, USA. Jul 9-13, 2007.

[26] R. Heckel and L. Mariani, "Automatic conformance testing of web services", *Fundamental Approaches to Software Engineering*, pp. 34-48, LNCS 3442, 2005.

[27] P. Mayer, "Design and Implementation of a Framework for Testing BPEL Compositions", *Master thesis, Leibniz University*, Hannover, Germany, Sep 2006.

[28] Z. Li, W. Sun, Z.B. Jiang, X. Zhang, "BPEL4WS Unit Testing: Framework and Implementation", *Proc of the IEEE International Conference on Web Service (ICWS'05)*, pp 103 - 110, 2005.