Journal of Ubiquitous Systems & Pervasive Networks
Volume 11, No. 1 (2019) pp. 01-09

Modeling and Verification of Response Time of QoS-aware Web Service Composition by Timed CSP

Ming Zhu, Xiaoliu Cui, Guodong Fan

College of Computer Science and Technology, Shandong University of Technology, Zibo, China

Abstract

Web service composition enables the provision/reusing of existing services in different business processes to satisfy different business requirements without investing in new infrastructure. QoS-aware web service composition seeks to help users find the optimal solution with maximization of users' satisfaction. A number of approaches based on Communicating Sequential Processes (CSP) have been proposed to model and verify properties of web service composition. However, little work has been done in verifying inputs, outputs and QoS criteria of web service composition. In this paper, we present a framework to model and verify QoS-aware web service composition by Timed CSP. It helps verify whether the service composition can accept inputs, generate outputs, and meet QoS requirements as specified. To do the verification, transformation rules that map QoS-aware web service composition to Timed CSP are defined. In order to explain the framework and transformation rules, we design a case study, where the model of QoS-aware web service composition is transformed to the model of process composition in Timed CSP and the program in machine-readable CSP (CSP_M). Furthermore, experiments are performed by using the Failure Divergence Refinement (FDR) tool to verify inputs, outputs, and QoS of the service composition.

Keywords: *QoS, service composition, Timed CSP, FDR.*

1. Introduction

The evolution of the Information and Communication Technology is an important factor of the explosion of Cloud Computing. The era of Cloud Computing raises the importance of web services. Web services are software modules that can be published, located and invoked on the web [1]. Web services provide a set of distributed computing resources like computing, application and storage by integrating Internet resources. As part of Service Computing, web services bring the evolution of development and deployment of distributed software. However, an individual service may fail to meet user's complicated requirement. Web Service Composition composes multiple web services together to fulfill complicated user requirement. To maximize user's satisfaction, researchers introduce Quality of Service (QoS) to web service composition. By using QoS criteria, it is possible to determine the usability and utility of a web service.

Recently, numerous formal approaches have been proposed to specify service compositions [2]. One of the major benefits of applying formal approaches is the possibility of verifying whether service compositions meet specific requirements and properties. Communicating Sequential Process (CSP) is a formal approach to specify concurrent systems [3], [4]. It has been suggested that CSP can be used to formally model and verify web services [5]. CSP makes it easy to specify and model message exchange between services, service composition, and other aspects [6].

Timed CSP is an extension to CSP by adding "real" time [7]. Timed CSP is the same language of CSP with the addition of a *WAIT* (*t*) statement that terminates successfully *t* time units after it has started. Roscoe introduced *tock*-CSP to verify discretely timed systems [8], where a special event *tock* represents the regular passage of time. Ouaknine theoretically connected *tock*-CSP to Timed CSP [9], which can translate Timed CSP into semantically equivalent *tock*-CSP. Timed CSP has been proved to be very successful in modelling and analyzing real-time concurrent system, and indeed has been used in numerous case studies [4], [10].

One of motivations of this work is to verify that, by providing inputs of a service composition, whether the composition can produce outputs with satisfied QoS in the specification. The other is that little research has been done on verifying QoS, specifically the response time, in web service composition based on CSP. Driven by the motivations, this paper aims to model and verify QoS-aware web service composition by Timed CSP. The key contributions of our work are as follows: a modeling and verification framework for QoS-aware web service composition is proposed, and transformation rules from QoS-aware web service composition to Timed CSP are defined. By using the framework and transformation rules, Timed CSP and the Failure Divergence Refinement (FDR) tool could verify whether inputs, outputs and QoS of service compositions conform to the requirements.

* Ming Zhu. Tel.: +86-15689080816
E-mail: zhu_ming@sdut.edu.cn ,

DOI: 10.5383/JUSPN.11.01.001

The rest of this paper is organized as follows. Section 2 and Section 3 describe preliminary knowledge and provide the background of this paper. The framework for modeling and verification of QoS-aware web service composition is given in Section 4. We present a case study and experimental results to illustrate the modeling and verification in Section 5. Section 6 reviews related work and the conclusion is drawn in Section 7.

2. Web Service Composition and QoS

This section introduces the definitions of web service, web service composition and QoS.

A web service w is defined as a tuple with the following components:

- w_{in} is a finite set of typed input parameters of w. A web service is invoked only when all its input parameters are satisfied.
- w_{out} is a finite set of typed output parameters of w.
- W_{QoS} is a finite set of quality-of-service (QoS) values of w. The criteria for QoS are determined from users' constraints and preferences.

A web service composition problem can be represented by a tuple with the following components:

- S is a finite set of services.
- C_{in} is a finite set of typed input parameters.
- C_{out} is a finite set of typed output parameters.
- C_{QoS} is a finite set of quality criteria.

We use plug-in matching degree to match services: two services can be connected if the input of a service is a subset of the output of the other service. This semantic model, borrowed from [11], is consistent with many proposed service composition approaches [12]-[14].

Services are connected either in sequence or in flow control. Services in sequence are invoked one by one ($w_1;w_2;\ldots;w_n$). Services in a flow control are invoked in parallel ($w_1\|w_2\|\ldots\|w_n$).

For web service compositions, there are several QoS criteria, such as response time, throughput, cost, and so on. For illustrative purpose, we focus on response time and the overall response time of service composition can be calculated as follows [1], [11]:

Response time (R): the interval between the receipt of an inquiry message and the beginning of the transmission of a response message.

$$R(w_1;w_2;\ldots;w_n)=\sum R(w_i) \quad (1)$$

$$R(w_1\|w_2\|\ldots\|w_n)=\max R(w_i) \quad (2)$$

3. Timed CSP

In this section, Timed CSP, machine-readable CSP (CSP_M), and Failure Divergence Refinement (FDR) tool for analyzing CSP are introduced.

3.1. Notations and Basic Concepts

Timed CSP terms are constructed according to the following grammar rules [9]:

$$P := STOP_t \mid SKIP_t \mid WAIT_t(n) \mid P_1 \overset{n}{\triangleright}_t P_2 \mid a \rightarrow_t P \mid P_1 \Box_t P_2 \mid P_1 \Pi_t P_2 \mid P_1 \|_B^t P_2 \mid P_1 \|\|_t P_2 \mid P_1;_t P_2 \mid \mu X.P \mid P_1 \lessdot_t \text{expr} \gtrdot_t P_2$$

These terms have the following intuitive interpretations:

- $STOP_t$ is the deadlocked, stable process which is only capable of letting time pass;
- $SKIP_t$ corresponds to the process $\surd \rightarrow_t$ STOP, where $\surd$ is a special final event that the process performs. It means a process, at any time, is willing to terminate successfully, and then do nothing.
- $WAIT_t(n)$ is the process which idles for n time units, and then becomes $SKIP$.
- $P_1 \overset{n}{\triangleright}_t P_2$ is the process that initially becomes P_1 for n time units, after which it silently becomes P_2 for no visible event occurs.
- $a \rightarrow_t P$ initially offers at any time to engage in the event a, and subsequently behaves like P.
- $P_1 \Box_t P_2$ denotes a process which is willing to have either like P_1 or P_2, at the choice of the environment.This decision is taken on the first visible event, and is nondeterministic only if this initial event is possible for both P_1 and P_2.
- $P_1 \Pi_t P_2$ represents the nondeterministic (or internal) choice between P_1 and P_2, which is independent of the environment.
- $P_1 \|_B^t P_2$ is a parallel composition of P_1 and P_2 over the interface set B. It means that P_1 and P_2 agree and synchronize on all events of set B, and to behave independently of each other with respect to all other events
- $P_1 \|\|_t P_2$ is an interleaving between P_1 and P_2, which means each process behaves independently of the other without synchronization.
- $P_1 ;_t P_2$ corresponds to the sequential composition of P_1 and P_2. It denotes a process which behaves like P_1 until P_1 chooses to terminate, at which point the process seamlessly starts to behave like P_2.
- $\mu X.P$ represents the unique solution to the equation $X=P$, where the variable X appears freely in P. The operator μX binds every free occurrence of X in P. The condition ensures that the recursion is well-defined and has a unique solution.
- $P_1 \lessdot_t expr \gtrdot_t P_2$ represents that if *expr* then behaves as P_1 else behaves as P_2.

A process can be defined as a set of *traces*. Each trace of a process is a finite sequence of symbols recording the events in which the process has engaged up to some moments in time [3]. For example, process P has a trace ‹ $event_1$, *tock*, $event_2$›, which indicates that P engages $event_1$ first, then time passes 1 unit (1 *tock*), then P engages $event_2$.

3.2. Failure Divergence Refinement (FDR) for Timed CSP

Failure Divergence Refinement (FDR) is a model checking tool for analyzing CSP systems [15]. FDR support both CSP and it's extension Timed CSP. To use FDR to model and verify timed CSP system, it requires programming in machine-readable CSP namely CSP_M , which combines the operators of CSP with a functional programming language. The latest version FDR 4.2.3 is released by Oxford University in Oct. 2017 [15]. In order to specify Timed CSP processes, CSP_M includes a *timed section* that automatically translates CSP processes to Timed CSP. Table 1 shows the mapping rules between terms for Timed CSP and CSP_M with *timed section*:

Table 1. Sample Mapping Rules Between Timed CSP And CSP_M

Terms of Timed CSP	Terms within timed section of CSP_M
STOPt	*STOP*
SKIPt	*SKIP*
WAITt(n)	*WAIT*(n)
$P_1 \triangleright_t P_2$	(P1 [] (WAIT(n); trig->P2))\trig;
$a \rightarrow_t P$	a->P
$P_1 \Box_t P_2$	P1 [] P2
$P_1 \|_B^t P_2$	P1 [|B|] P2
$P_1 \|\|\|_t P_2$	P1|||P2
$P_1;_t P_2$	P1; P2
$P_1 \lhd_t expr \rhd_t P_2$	If *expr* then P1 else P2

To illustrate the translation between Timed CSP and CSP_M , we now introduce an example. Given a process P that first performs an event *start*, then waits 5 time units, and finally becomes P. P can be described in Timed CSP as $P = start \rightarrow_t WAIT(5); P$. According to Table 1, the representation of Timed CSP of P is able to be translated into a CSP_M program as follows:

```
channel tock,start
AllZero(_) = 0
Timed(AllZero) {
    P = start -> WAIT(5);P
```

In CSP_M, the key word *channel* is used to define events. There are two channels declared, *tock* and *start*. As stated before, *tock* is a special event to represent a time unit. *start* is an event that process *P* can accept. *AllZero*() is a function that defines the execution of each event in the timed section costs 0 time unit. *Timed*(AllZero) with a pair of curly brackets defines the scope of timed section, where all the declarations are translated to Timed CSP. *P = start ->WAIT*(5); *P* is the CSP_M translation of Timed CSP $P = a \rightarrow_t WAIT_t(5);P$.

4. The Framework For Modeling And Verification Of QoS-aware(response time) Web Service Composition

In this section, we establish the framework for modeling and verification of QoS-aware (response time) web service composition by Timed CSP, and then we define the transformation rules used in the framework for mapping QoS-aware (response time) web service composition to Timed CSP.

4.1. The Framework for Modeling and Verification

The framework consists of the following steps (Figure 1).

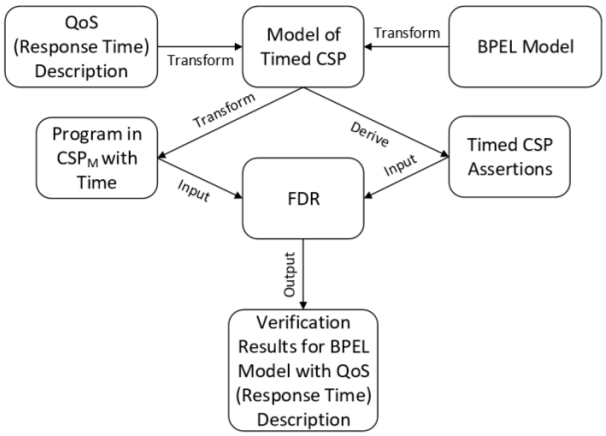

Fig.1. The Framework

1) Transform the models of BPEL with QoS (response time) descrpition to the models of Timed CSP.
2) Transform the models of Timed CSP to programs in CSP_M with timed section.
3) Derive CSP_M assertions based on the models of Timed CSP and the requirements of BPEL with QoS (response time) description.
4) Use FDR to verify the programs in CSPM against the assertions, and generate verification results.

4.2. Transformation from Models of BPEL with QoS(response time) to the models of Timed CSP and CSP_M

To transform the models of BPEL with QoS (response time) description to the models of Timed CSP, corresponding transformation rules needs to be defined. In PBEL V2.0, the major building blocks of BPEL processes are activities, which include basic activities and structured activities [16]. As BPEL orchestrations in composing services do not evaluate the QoS properties of specific services, we associated QoS properties of services with BPEL activities using them. In this section, we detail our formulations to QoS (response time) for some basic and structured BPEL activities.

Basic Activities. In research [17], it suggests that all basic activities except invoke complete instantaneously when they start, which don't need to be analyzed with QoS properties. In this paper, we focus on response time for invoke activity. The invoke activity is used to call a web service provided by a partner. An invoke activity includes two type, *one-way* activity and *request-response* activity. The QoS (response time) for the invoke activity is specified using the Web Service Level Agreements (WSLA).

1. one-way invoke

An *one-way invoke* activity requires an input variable, then executes with the process logic without waiting for the reply. Given an one-way activity *service*A in BPEL along with its QoS (response time) *10* in WSLA as follows:

```
<invoke name="serviceA"
    operation="serviceAOperation"
    inputVariable="Input" />

<SLAParameter>
      AverageResponseTimeServiceA
</SLAParameter>
  <Value > 10 </ Value >
```

A one-way invoke activity of BPEL can be transformed into a process with an *Input* action along with time section $WAIT_t$ in Timed CSP. This process first accepts *Input* through channel *service*A*Operation*, then it waits for *10* time units which represents the QoS (response time), and end successfully. The CSP transformation is show as follows:

$$service\text{A}Operation?Input \rightarrow_t WAIT_t(10) \rightarrow_t SKIP_t$$

The *one-way invoke* activity is represented as follows in CSP_M:

serviceAOperation?Input -> WAIT(10) -> SKIP

2. request-response invoke

A *request-response invoke* activity requires both an input and output variable, and it would block the process until it receives a response from the partner service. Given a request-response activity *service*B in BPEL along with it's QoS (response time) *t* in WSLA as follows:

```
<invoke name="serviceB"
    operation="serviceBOperation"
    inputVariable="Input"
    outputVariable="Output" />

<SLAParameter>
      AverageResponseTimeServiceB
</SLAParameter>
   <Value > 10 </ Value >
```

A request-response invoke activity of BPEL can be transformed into a process with an *Input* action and an *Output* action along with time section $WAIT_t$ in Timed CSP. This process first accepts *Input* through channel *service*B*Operation*, then it waits for *10* time units which represents the QoS (response time), generates *Output* through channel *service*B*Operation*, and ends successfully. The CSP transformation is show as follows:

$$service\text{B}Operation?Input \rightarrow_t \text{WAIT}_t(10);$$
$$service\text{B}Operation!Output \text{ -> } \text{SKIP}_t$$

The *request-response invoke* activity is represented as follows in CSP_M :

serviceBOperation?Input -> WAIT(10);
serviceBOperation!Output -> SKIP

Structured Activities. This kind of activities can contain other activities and define the business logic between them. In this paper, we discuss the transformation from *sequence*, *if-else*, *while*, *pick* and *flow* activities to the corresponding Timed CSP representations respectively.

1. sequence

A *sequence* structured activity is used to define a collection of activities which are executed sequentially in predefined order. Given a sequence structured activity in BPEL as follows:

```
<sequence name=...>
    <...activity1.../>
    <...activity2.../>
        ...
    <...activityN.../>
</sequence>
```

For each $activity_i$ ($1 \le i \le$ N)inside *sequence*, given that the corresponding QoS (response time) is t_i. In the process of the transformation, the above-mentioned *sequence* structured activity can be transformed to a sequence of processes in Timed CSP, where each $activity_i$ with QoS (response time) t_i can be transformed to a process P_i with corresponding waiting time t_i, and the order between activities is as same as the order between corresponding processes. The CSP transformation is show as follows:

$$P_1;_t WAIT_t(t_1);$$
$$P_2;_t WAIT_t(t_2);$$
$$...;_t$$
$$P_N;_t WAIT_t(t_N)$$

The *sequence* activity is represented as follows in CSP_M :

P1;WAIT(t1);
P2;WAIT(t2);
...;
PN;WAIT(tN)

2. if-else

A *if-else* structured activity allows exactly one choice of activity from a given set of choices to be selected. For each choice, the behavior is to check a condition and if that condition evaluates to true, the associated branch is executed, otherwise an alternative path is taken. Given a *if-else* structured activity in BPEL as follows:

```
<if name="choices">
    <condition>
        condition1
    </condition>
    <...activity1.../>
    <elseif>
        <condition>
            condition2
        </condition>
        <...activity2... />
    </elseif>
    <else>
        <...activity3 ... />
    </else>
</if>
```

For each $activity_i$ ($1 \le i \le 3$) inside *if-else*, given that the corresponding QoS (response time) is t_i. In the process of the transformation, the above-mentioned *if-else* structure can be transformed to *conditional choices* in Timed CSP, where each $activity_i$ with QoS (response time) t_i can be transformed to a process P_i with corresponding waiting time t_i, and a $condition_j$ ($1 \le j \le 2$) can be transformed to a conditional expression $expr_j$. The CSP transformation is show as follows:

$$(p_1;_t WAIT_t(t_1))$$
$$\trianglelefteq expr_1 \trianglerighteq$$
$$(((p_2;_t WAIT_t(t_2))$$
$$\trianglelefteq expr_2 \trianglerighteq$$
$$(p_3;_t WAIT_t(t_3)))$$

The *if-else* activity is represented as follows in CSP_M :

```
if expr1
    then P1;WAIT(t1)
    else if expr2
        then P2;WAIT(t2)
        else P3;WAIT(t3)
```

3. while

A *while* structured activity has a child activity nested within. It allows the child activity to be executed repeatedly as long as a given condition evaluates to true. The condition is specified on the while activity and gets evaluated at the beginning of each iteration. Given a *while* structured activity in BPEL as follows:

```
<while>
    <condition>
        condition
```

```
    </condition>
    <...activity ... />
</while>
```

For the *activity* inside *while*, given that the corresponding QoS (response time) is *t*. In the transformation, the abovementioned *while* structured activity can be transformed to a *recursion* with a *conditional choice* in Timed CSP. If the *conditional choice* is true, the recursion continues. Otherwise, the recursion ends successfully. The *activity* can be transformed to a process *P* with corresponding waiting time *t*, and the *condition* can be transformed to a conditional expression *expr*. The CSP transformation is show as follows:

$$\mu P.((P;WAIT_t(t)) \lhd_t expr \rhd_t SKIP_t)$$

The *while* activity is represented as follows in CSP_M:

```
X = if expr
    then P;WAIT(t);X
    else SKIP
```

4. pick

A *pick* structured activity has includes several *onMessage* elements, one of which will be triggered to execute after the structure receives the corresponding message. Each onMessage element points to an activity and to a variable that holds the received message. Given a *pick* structured activity in BPEL as follows:

```
<pick>
    <onMessage ...
        ...activity1...
        variable="var1">
    </onMessage>
    <onMessage ...
        ...activity2...
        variable="var2">
    </onMessage>
    ...
    <onMessage ...
        ...activityN...
        variable="varN">
    </onMessage>
</pick>
```

For each $activity_i$ ($1 \le i \le N$) inside *pick*, given that the corresponding QoS (response time) is t_i. In the transformation, the above-mentioned *pick* structured activity can be transformed to *deterministic choices* in Timed CSP, where each $activity_i$ with QoS (response time) t_i can be transformed to a process P_i with corresponding waiting time t_i, and the relationship between $activity_i$ can be transformed to *deterministic choices* between P_i. The CSP transformation is show as follows:

$$(P_1;_t WAIT_t(t_1)) \,\Box_t$$
$$(P_2;_t WAIT_t(t_2)) \,\Box_t$$
$$\ldots \,\Box_t$$
$$(P_N;_t WAIT_t(t_N))$$

The *pick* activity is represented as follows in CSP_M:

```
(P1;WAIT(t1)) [ ]
(P2;WAIT(t2)) [ ]
... [ ]
(PN;WAIT(tN))
```

5. flow

A *flow* structured activity allows child activities to be executed in parallel. Given a *flow* structured activity in BPEL as follows:

```
<flow ...>
    <...activity1... />
    <...activity2... />
    ...
    <...activityN... />
</flow>
```

For each $activity_i$ ($1 \le i \le N$)inside *flow*, given that the corresponding QoS (response time) is t_i. In the process of the transformation, the above-mentioned *flow* structured activity can be transformed to *parallel comosition* in Timed CSP, where each $activity_i$ with QoS (response time) t_i can be transformed to a process P_i with corresponding waiting time t_i., and the relationship between $activity_i$ can be transformed to *parallel* between P_i. If it is necessary to synchronize between some of these activities, the synchronized information can be represented as a set of events associated with *parallel*. The CSP transformation is show as follows:

$$(P_1; WAIT(t_1)) \|_t$$
$$(P_2; WAIT(t_2)) \|_t$$
$$\ldots \|_t$$
$$(P_N ; WAIT(t_N))$$

The *flow* activity is represented as follows in CSP_M :

```
(P1;WAIT(t1))||
(P2;WAIT(t2))||
...||
(PN;WAIT(tN))
```

5. Case Study

In this section, we introduce a case study to illustrate the application of the framework and the transformation rules for modeling and verifying inputs, outputs, and QoS of web service composition.

5.1. A Web Service Composition

Let's consider an online booking system for the travelling agency to book tickets and hotels. The system contains has four collaborative web services: *AgReq*, *AgRcv*, *Al* and *Htl*. The travelling agency uses *AgReq* to send service requests, and uses *AgRcv* to receive service responses. *Al* is used for booking a flight with an airline, and *Htl* is used for booking a room in a hotel. The services information is shown in Table 2. The service composition example is shown in Figure 2.

Table 2. Services Information

Service	Inputs	Outputs	R[1]
AgReq	flightInfo,horeInfo	agFlight,agHotel	2
Al	agFlight	flightResult	4
Htl	agHotel	hotelResult	3
AgRcv	flightResult,hotelResult	result	1

[1] R:response time(ms)

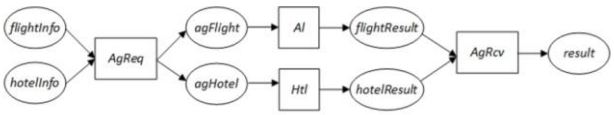

Fig.2. The Online Booking System

5.2. Timed CSP Model for QoS-aware Web Service Composition

When QoS-aware web service compositions are obtained, we can construct the corresponding Timed CSP models.

Service AgReq is represented as the process *AgReqP*.

$$AgReqP = AgReqPIn;_t AgReqPOut$$

$$AgReqPIn = flightInfo \rightarrow_t hotelInfo \rightarrow_t AgReqPQoS$$
$$\Box_t hotelInfo \rightarrow_t flightInfo \rightarrow_t AgReqPQoS$$

$$AgReqPQoS = WAIT_t(2)$$

$$AgReqPOut = agFlight \rightarrow_t agHotel \rightarrow_t AgReqP$$
$$\Box_t agHotel \rightarrow_t agflight \rightarrow_t AgReqP$$

Service *Al* is represented as the process *AlP*.

$$Alp = AlPIn;_t AlPOut$$

$$AlPIn = agFlight \rightarrow_t AlPQoS$$

$$AlPQoS = WAIT_t(4)$$

$$AlPOut = flightResult \rightarrow_t AlP$$

Service *Htl* is represented as the process *HtlP*.

$$HtlP = HtlPIn;_t HtlPOut$$

$$HtlPIn = agHotel \rightarrow_t HtlPQoS$$

$$HtlPQoS = WAIT_t(3)$$

$$HtlPOut = hotelResult \rightarrow_t HtlP$$

Service *AgRcv* is represented as the process *AgRcvP*.

$$AgRcvP = AgRcvPIn;_t AgRcvPOut$$

$$AgRcvPIn = flightResult \rightarrow_t hotelResult \rightarrow_t AgRcvPQoS$$
$$\Box_t hotelResult \rightarrow_t flightResult \rightarrow_t AgRcvPQoS$$

$$AgRcvPQoS = WAIT_t(1)$$

$$AgRcvPOut = result \rightarrow_t AgRcvP$$

The web service compositions *Al*||*Htl*, *AgReq*; (*Al*||*Htl*), and *AgReq*; (*Al*||*Htl*); *AgRcv* can be represented as process compositions as follows:

- Service composition *Al*||*Htl* is a represented as the process composition $AlHtl=AlP|||_t HtlP$.
- Servide composition *AgReq*;(*Al*||*Htl*) is represented as the process composition $AgReqAlHtl = AgReqP \underset{\{agFlight,agHotel\}}{||_t} AlHtl$.
- Service composition *AgReq*;(*Al*||*Htl*);*AgRcv* is represent-ted as the process composition AgReqAlHtl *AgRcv*= $AgReqAlHtl \underset{\{flightResult,hotelResult\}}{||_t} AgRcv$.

Based on the Timed CSP representations of web services and service compositions, we can build a CSP_M program as follows:

```
channel tock,flightInfo,hotelInfo,
channel agFlight,agHotel
channel flightResult,hotelResult
channel result


AllZero(_)=0

Timed(AllZero){
AgReqP=AgReqPIn;AgReqPOut
AgReqPIn=hotelInfo->flightInfo->AgReqPQos
            []flightInfo->hotelInfo->AgReqPQos
AgReqPQos=WAIT(2)
AgReqPOut=agFlight->agHotel->AgReqP
            []agHotel->agFlight->AgReqP

AgRcvP=AgRcvPIn;AgRcvPOut
AgRcvPIn=flightResult->hotelResult->AgRcvPQos
            []hotelResult->flightResult->AgRcvPQos
AgRcvPQos=WAIT(1)
AgRcvPOut=result->AgRcvP

AlP=AlPIn;AlPOut
AlPIn=agFlight->AlPQos
AlPQos=WAIT(4)
AlPOut=flightResult->AlP

HtlP=HtlPIn;HtlPOut
HtlPIn=agHotel->HtlPQos
HtlPQos=WAIT(3)
HtlPOut=hotelResult->HtlP

AlHtl=AlP|||HtlP
AgReqAlHtl=AgReqP[|{agFlight,agHotel}|]AlHtl
AgReqAlHtlAgRcv=AgReqAlHtl
    [|{hotelResult,flightResult}|]AgRcvP
```

5.3. Experimental Evaluation

Based on the CSP_M code generated previously, FDR can use the *Has Trace Assertions* to verify inputs, outputs, and QoS (response time) of processes and process compositions that represent the corresponding web services and web service compositions respectively.

Our experiments are implemented on a personal computer with a Windows 10 64-bit operating system, 8 GB RAM memory, Intel Core i5-7200U processor. The version of FDR used in the experiments is 4.2.3 (Windows 64-bit) released on 26/10/2017.

For each process or composition implemented in CSP_M , FDR can generate a transition diagram for it to help understand its behaviors. For example, the transition diagrams of *AgReqP* and *AgRcvP* are illustrated in Figure 3 and Figure 4 respectively.

We build a trace in Timed CSP and use *Has Trace Assertions* to verify the process composition *AgReqAlHtlAgRcv*.

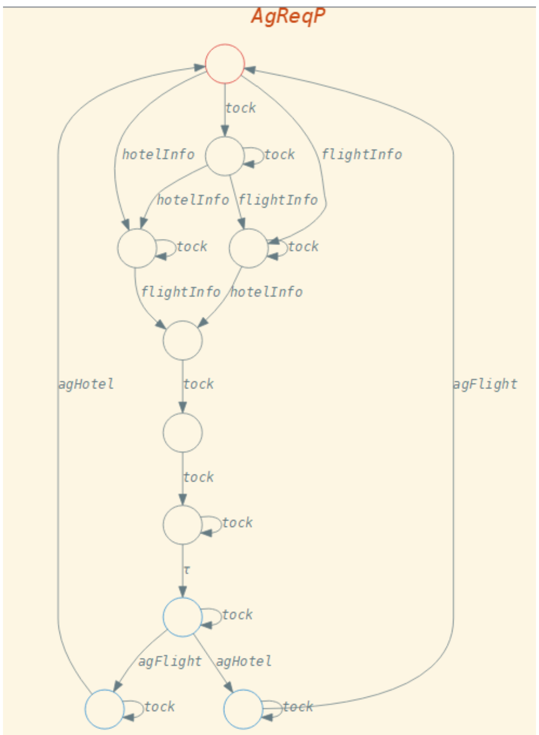

Fig.3. The Transition Diagram of *AgReqP*

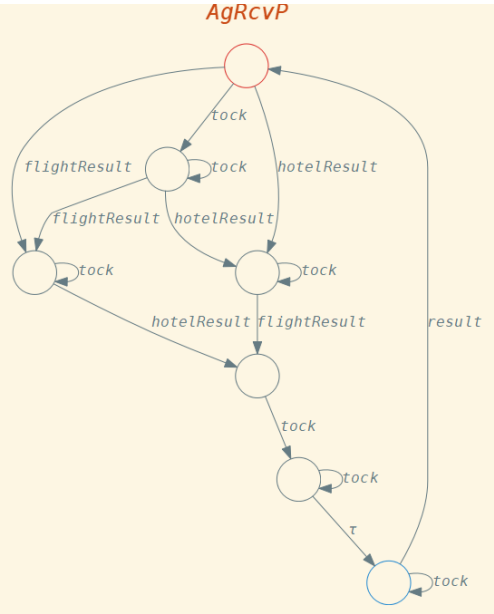

Fig.4. The Transition Diagram of *AgRcv*

- We construct a trace *T* according to the specification of process composition *AgReqAlHtlAgRcv* and its corresponding web service composition, ‹ *flightInfo*, *hotelInfo*, *tock*, *tock*, *agFlight*, *agHotel*, *tock*, *tock*, *tock*, *hotelResult*, *tock*, *flightResult*, *tock*, *result* ›. This trace means *flightInfo* and *hotelInfo* occur at the beginning; after 2 time units (2 *tocks*), messages *agFlight* and *agHotel* occur; then, time passes 3 units (3 *tocks*); after that, message *hotelResult* appears; 1 time unit (1 *tock*) later, message *flightResult* occurs; and after 1 time unit (1 *tock*), message *result* is generated.
- We make an assertion for *AgReqAlHtlAgRcv* in FDR based on the trace *T*, assert AgReqAlHtlAgRcv :[has trace]: < flightInfo, hotelInfo, tock, tock, agFlight, agHotel, tock, tock, tock, hotelResult, tock, flightResult, tock, result >.

Figure 5 from FDR shows that the assertion passes, and the verification finishes in 0.11s. This indicates that by offering *flightInfo* and *hotelInfo* as the input messages, the composition can produce *result* in 7 time units. The result of executing the assertion in FDR confirms that with the specified inputs, the web service composition *AgReq*; (*Al*||*Htl*); *AgRcv* can generate specified outputs with QoS response time satisfied.

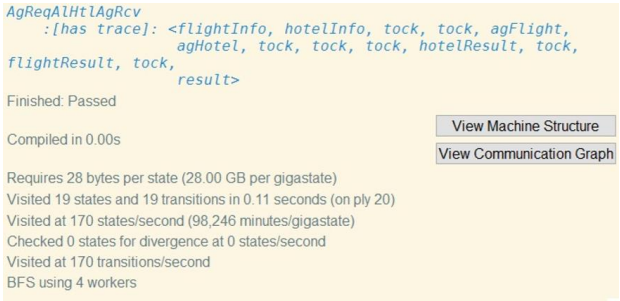

Fig.5. The Verification Result of Original Composition *AgReqAlHtlAgRcv*

We give a counterexample to show that a web service composition fails to generate outputs within specified response time. Let's modify the response time of service *Al* to 6, while the travelling agency still expect the service's response time to be 4. By following the framework and transformation rules, the modified service composition can be transformed to a process composition with process *AlP* waiting 6 time units, and the corresponding program in CSP_M can be generated. The assertion used previously can represent the travelling agency's expectation. When using FDR to verify the program against the assertion, Figure 6 from FDR shows that the assertion fails this time, and the verification finishes in 0.13s. This indicates that by offering *flightInfo* and *hotelInfo* as the inputs, the composition cannot produce *result* as output within the expected response time. It fails to meet the requirements of inputs, outputs, and QoS of the travelling agency.

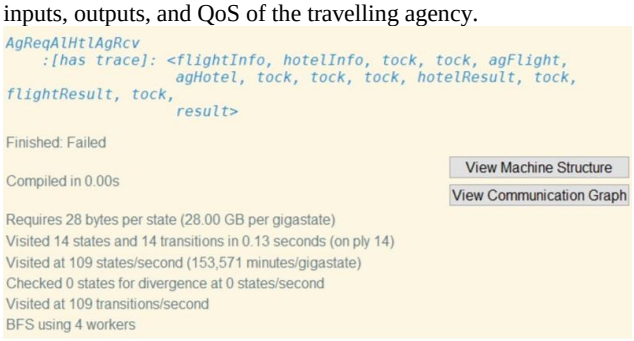

Fig. 6. The Verification Result of Original Composition *AgReqAl-HtlAgRcv* with WAIT(6)in *AlP*

By using FDR's debug functionality, we can further examine the reason of the assertion failure. Figure 7 shows that the event *flightResult* cannot be generated in 4 time units in process *AlP* after accepting the event *agFlight*, while the assertion requires the event *agFlight* to be generated in 4 time units.

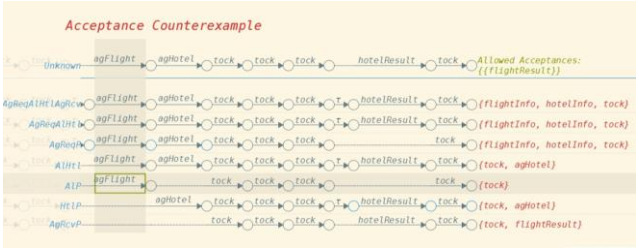

Fig. 7. Counterexample Exhibition

6. Related Work

The goal of QoS-aware service composition is to select competitive services and optimize the whole QoS value. In this process, optimization algorithms are utilized to reduce the search space, minimize the search time and enhance accuracy of obtained solutions. Paper [18] presents a hybrid artificial bee colony (HABC) algorithm to solve the cloud manufacturing service optimal selection problem. In this process, chaotic ergodic search is applied to avoid premature of the algorithm. In our previous work, we propose the application of a skyline operator to reduce the search space and improve the scalability. We also present a partial pre-composing approach which stores popular paths for fast delivery [14]. Zheng et al. demonstrate that QoS values are not the same to different consumers even when calling the same service [19]. Under this consideration, QoS prediction approaches are proposed [20], [21]. Research [22] proposes a local search enhanced hybrid artificial bee colony algorithm (HABC) for solving the multi-objective flexible task scheduling problem in Cloud computing system.

In recent years, there are a number of initiatives tend to use the CSP to model and verify web service composition. S.

Ripon *et al*. combine compensating CSP and Finite State Process to handle faults in long running transactions in web services, and verify properties of composition and execution of compensations [6]. Xu et al. use CSP to formalize the specifications based on web service choreography description language (CDL) [23]. Y. Zhu *et al*. propose a framework that transform Business Process Execution Language (BPEL) to CSP, and use FDR to verify deadlock, liveness, and safety of compositions [24]. Most of the work focuses on verifying the liveness, safety, deadlock and other properties of service composition, while little research has been done in verifying QoS of service composition.

7. Conclusion

This paper proposes a framework by using Timed CSP based on model transformation. Firstly, the modeling and verification framework between QoS-aware web service composition and Timed CSP is introduced. Secondly, transformations rules from QoS-aware Web Service Composition to Timed CSP are defined in details. Thirdly, to illustrate the framework and the transformation rules, a case study of online booking system is developed. In doing so, it shows that models of QoS-aware service composition can be transformed to models of process composition in Timed CSP and programs in CSPM with timed section. By using the framework and transformation rules, the approach verifies whether inputs, outputs and QoS of service compositions conform to the requirements. Furthermore, the experiments indicate that inputs, outputs and response time in web service compositions can be verified by the tool FDR with CSP_M assertions. In future, more details on verification of QoS-aware web service composition and more case studies will be discussed.

References

[1] M. Papazoglou, Web Services: Principles and Technology. Prentice Hall, 2011.

[2] G. M. M. Campos, N. S. Rosa, and L. F. Pires, "A survey of formalization approaches to service composition," in Services Computing (SCC), 2014 IEEE International Conference on, June 2014, pp. 179–186. https://doi.org/10.1109/SCC.2014.32

[3] C. A. R. Hoare, Communicating sequential processes. Englewood Cliffs, United States: Prentice-Hall, 1985. Proceedings of the 1st MIT conference on CFSM. Cambridge, MA, 2001.

[4] A. W. Roscoe, Understanding concurrent systems. London, United Kingdom: Springer, 2010. https://doi.org/10.1007/978-1-84882-258-0

[5] G. Salaun, L. Bordeaux, and M. Schaerf, "Describing and reasoning on web services using process algebra," in Proceedings. IEEE International Conference on Web Services, July 2004, pp. 43–50. https://doi.org/10.1109/ICWS.2004.1314722

[6] S. Ripon, F. Sultana, and F. Rahman, "Verification of service composition and compensation by using process algebra," Journal of Advances in Computer Networks, vol. 4, no. 4, pp. 193–200, 2016.

[7] G. Reed and A. Roscoe, "A timed model for communicating sequential processes," Theoretical Computer Science, vol. 58, no. 1-3, pp. 249-261, 1988. https://doi.org/10.1016/0304-3975(88)90030-8

[8] A. W. Roscoe, The Theory and Practice of Concurrency. Prentice Hall, 1998.

[9] J. Ouaknine, "Discrete analysis of continuous behaviour in real-time concurrent systems," Ph.D. dissertation, University of Oxford, Michaelmas, United Kingdom, 2000.

[10] J. Ouaknine and S. Schneider, "Timed csp: A retrospective," Electronic Notes in Theoretical Computer Science, vol. 162, pp. 273-276, 2006. https://doi.org/10.1016/j.entcs.2005.12.093

[11] S. Bleul, T. Weise, and K. Geihs, "The web service challenge - a review on semantic web service composition," Electronic Communications of the EASST, vol. 17, 2008.

[12] W. Jiang, C. Zhang, Z. Huang, M. Chen, S. Hu, and Z. Liu, "QSynth: A tool for QoS-aware automatic service compo-sition," in Web Services (ICWS), 2010 IEEE International Conference on, July 2010, pp. 42-49. https://doi.org/10.1109/ICWS.2010.38

[13] P. Rodriguez-Mier, M. Mucientes, and M. Lama, "A dynamic QoS-aware semantic web service composition algorithm," in Service-Oriented Computing, ser. Lecture Notes in Computer Science, C. Liu, H. Ludwig, F. Toumani, and Q. Yu, Eds. Springer Berlin Heidelberg, 2012, vol. 7636, pp. 623-630. https://doi.org/10.1007/978-3-642-34321-6_48

[14] J. Li, Y. Yan, and D. Lemire, "Scaling up web service composition with the skyline operator," in IEEE International Conference on Web Services, 2016, pp. 147–154. https://doi.org/10.1109/ICWS.2016.27

[15] FDR manual. https://www.cs.ox.ac.uk/projects/fdr/manual/ cited[12/25/2018].

[16] Web services business process execution language. https://www.oasis-open.org/committees/tc_home.php?wg abbrev=wsbpel cited[2/25/2018].

[17] D. Mukherjee, P. Jalote, and M. Nanda, "Determining qos of ws-bpel compositions," in International Conference on Service-Oriented Computing 2008. Germany: Springer Berlin, 2008, pp. 378–393. https://doi.org/10.1007/978-3-540-89652-4_29

[18] J. Zhou and X. Yao, "A hybrid artificial bee colony algorithm for optimal selection of qos-based cloud manufacturing service composition," The International Journal of Advanced Manufacturing Technology, vol. 88, no. 9, pp. 3371-3387, Feb 2017. [Online]. Available: https://doi.org/10.1007/ s00170-016-9034-1. https://doi.org/10.1007/s00170-016-9034-1

[19] Z. Zheng, Y. Zhang, and M. Lyu, "Investigating QoS of real-world web services," Services Computing, IEEE Transactions on, vol. 7, no. 1, pp. 32-39, Jan 2014. https://doi.org/10.1109/TSC.2012.34

[20] H. Wang, H. Sun, and Q. Yu, "Reliable service composition via automatic qos prediction," in Proceedings -IEEE 10t International Conference on Services Computing, SCC 2013, 06 2013, pp. 200-207. https://doi.org/10.1109/SCC.2013.45

[21] R. Karim, C. Ding, and A. Miri, "End-to-end QoS prediction of vertical service composition in the cloud," in Cloud Com-puting (CLOUD), 2015 IEEE 8th International Conference on, June 2015, pp. 229-236. https://doi.org/10.1109/CLOUD.2015.39

[22] J. Li, Y. Han, C. Wang, "A Hybrid Artificial Bee Colony Algorithm to Solve Multi-objective Hybrid Flowshop in Cloud Computing Systems," in The 3rd International Conference on Cloud Computing and Security, Nanjing, China, pp. 68-73, 2017 https://doi.org/10.1007/978-3-319-68505-2_18

[23] D. Xu, Z. Lei, W. Li, and B. Zhang, "Model checking web services choreography in process analysis toolkit," Journal of Shanghai University (English Edition), vol. 14, no. 1, pp. 45-49, Feb 2010. https://doi.org/10.1007/s11741-010-0109-3

[24] Y. Zhu, Z. Huang, and H. Zhou, "Modeling and verification of web services composition based on model transformation," Journal of software practice and experience, vol. 47, no. 5, pp. 709-730, 2016. https://doi.org/10.1002/spe.2434