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(CSPm). 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 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 \( \text{WAIT} (t) \) statement that terminates successfully \( t \) time units after it has started. Roscoe introduced \( \text{ock-CSP} \) to verify discretely timed systems [8], where a special event \( \text{ock} \) represents the regular passage of time. Ouaknine theoretically connected \( \text{ock-CSP} \) to Timed CSP [9], which can translate Timed CSP into semantically equivalent \( \text{ock-CSP} \). 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@sduet.edu.cn
© 2019 International Association for Sharing Knowledge and Sustainability.
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 \((w_{1}[w_{2}], \ldots, [w_{m}])\).

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]:

\[
\text{Response time (R): the interval between the receipt of an inquiry message and the beginning of the transmission of a response message.}
\]

\[ R(w_{1},w_{2},\ldots,w_{n}) = \sum R(w_{i}) \] (1)

\[ R(w_{1}[w_{2}],\ldots,[w_{m}]) = \max R(w_{i}) \] (2)

3. Timed CSP

In this section, Timed CSP, machine-readable CSP (CSP\text{\textregistered}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]:

\[
P \rightarrow \text{STOP}_P \mid \text{SKIP}_P \mid \text{WAIT}(...,n) \mid \text{P}_1 \overset{a}{\rightarrow} \text{P}_2 \mid P_1 \parallel P_2 \mid P_1 \parallel \text{P}_2 \mid I \parallel \text{P}_2 \mid P_1 \parallel I \parallel \text{P}_2 \mid P_1 \parallel P_2 \parallel \mu X . P
\]

\[
P_1 \prec \text{expr} \rightarrow \text{P}_2
\]

These terms have the following intuitive interpretations:

- \( \text{STOP}_P \): is the deadlocked, stable process which is only capable of letting time pass;
- \( \text{SKIP}_P \): corresponds to the process \( \varnothing \rightarrow \text{STOP} \), where \( \varnothing \) 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.
- \( \text{WAIT}(n) \): is the process which idles for \( n \) time units, and then becomes \( \text{SKIP} \).
- \( P_1 \parallel P_2 \): 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 \( \rightarrow \): \( P \) initially offers at any time to engage in the event \( a \), and subsequently behaves like \( P \).

\( P_1 \parallel \text{P}_2 \): 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 \( P_2 \).

\( P_1 \parallel \text{P}_2 \): represents the nondeterministic (or internal) choice between \( P_1 \) and \( P_2 \), which is independent of the environment.

\( P_1 \parallel \text{P}_2 \): 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.

\( P_1 \parallel \text{P}_2 \): an interleaving between \( P_1 \) and \( P_2 \), which means each process behaves independently of the other without synchronization.

\( P_1 \parallel \text{P}_2 \): 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 \( \mu X \) binds every free occurrence of \( X \) in \( P \). The condition ensures that the recursion is well-defined and has a unique solution.

\( P_1 \prec \text{expr} \rightarrow \text{P}_2 \): represents that if \( \text{expr} \) 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 \( \langle \text{event}_1, \text{tick}, \text{event}_2 \rangle \), which indicates that \( P \) engages \( \text{event}_1 \), first, then time passes 1 unit (1 tick), then \( P \) engages \( \text{event}_2 \).

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 its’ extension Timed CSP. To use FDR to model and verify timed CSP system, it requires programming in machine-readable CSP namely CSP\text{\textregistered}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\text{\textregistered}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\text{\textregistered}M with timed section:
Properties. In M-valuate the QoS-imed CSP as S-ed MP

1. **Basic Activities.** To illustrate the translation between Timed CSP and CSPM, 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 \( P = \text{start} \rightarrow \text{WAIT}(5); P \). According to Table 1, the representation of Timed CSP of P is able to be translated into a CSPM program as follows:

\[
\begin{align*}
\text{channel} & \text{ tock, start} \\
\text{AllZero(\_)} & = 0 \\
\text{Timed}(\text{AllZero}) & \{ \\
& \text{P} = \text{start} \rightarrow \text{WAIT}(5); P \\
\}
\end{align*}
\]

In CSPM, 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. \( P = \text{start} \rightarrow \text{WAIT}(5); P \) is the CSPM translation of Timed CSP \( P = \text{start} \rightarrow \text{WAIT}(5); P \).

2. **Transformation from Models of BPEL with QoS(response time) to the models of Timed CSP and CSPM.** 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 serviceA in BPEL along with its QoS (response time) 10 in WSLA as follows:

```xml
<invoke name="serviceA"
operation="serviceAOperation"
inputVariable="Input" />
<SLAParameter>
AverageResponseTimeServiceA
</SLAParameter>
<Value> 10 </Value>
```

**Fig.1. The Framework**

1. **Transform the models of BPEL with QoS (response time) description to the models of Timed CSP.**
2. **Transform the models of Timed CSP to programs in CSPM with timed section.**
3. **Derive CSPM 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.**

### Table 1. Sample Mapping Rules Between Timed CSP and CSPM

<table>
<thead>
<tr>
<th>Terms of Timed CSP</th>
<th>Terms within timed section of CSPM</th>
</tr>
</thead>
<tbody>
<tr>
<td>STOP</td>
<td>STOP</td>
</tr>
<tr>
<td>SKIP</td>
<td>SKIP</td>
</tr>
<tr>
<td>WAIT(n)</td>
<td>WAIT(n)</td>
</tr>
<tr>
<td>( P_1 \rightarrow P_2 )</td>
<td>(P1 (</td>
</tr>
<tr>
<td>( a \rightarrow P )</td>
<td>a( \rightarrow P )</td>
</tr>
<tr>
<td>( P_1 | P_2 )</td>
<td>P1 (</td>
</tr>
<tr>
<td>( P_1 [</td>
<td>B</td>
</tr>
<tr>
<td>( P_1 &amp;&amp; P_2 )</td>
<td>P1 &amp;&amp; P2</td>
</tr>
<tr>
<td>( P_1 &amp;&amp; \text{expr} )</td>
<td>If \text{expr} then P1 else P2</td>
</tr>
</tbody>
</table>


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

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

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

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

### 2. request-response invoke

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

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

```
<SLAPParameter>
    AverageResponseTimeServiceB
</SLAPParameter>
```

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, in Timed CSP. This process first accepts Input through channel serviceBOperation, then it waits for 10 time units which represents the QoS (response time), generates Output through channel serviceBOperation, and ends successfully. The CSP transformation is show as follows:

```
serviceBOperation?Input -> WAIT(10);
```

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

```
serviceBOperation?Input -> WAIT(10);
```

### 3. Sequence

**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, (1 ≤ i ≤ N) inside sequence, given that the corresponding QoS (response time) is ti. 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, with QoS (response time) ti can be transformed to a process Pi with corresponding waiting time ti, and the order between activities is as same as the order between corresponding processes. The CSP transformation is show as follows:

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

The sequence activity is represented as follows in CSPM:

```
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>
  <else>
    <condition>
      condition2
    </condition>
    <else>
      <activity1> ...
    </else>
  </else>
</if>
```

```
<SLAPParameter>
    AverageResponseTimeServiceB
</SLAPParameter>
```

For each activity, (1 ≤ i ≤ 3) inside if-else, given that the corresponding QoS (response time) is ti. In the process of the transformation, the above-mentioned if-else structure can be transformed to conditional choices in Timed CSP, where each activity: with QoS (response time) ti can be transformed to a process Pi with corresponding waiting time ti, and a condition (1 ≤ j ≤ 2) can be transformed to a conditional expression expr.

The CSP transformation is show as follows:

```
(\(p_i; \text{WAIT}(t_i)\))
\[\text{if} \ expr1 \ \text{then} \ P1; \text{WAIT}(t1) \ \\
\text{else if} \ expr2 \ \text{then} \ P2; \text{WAIT}(t2) \ \\
\text{else} \ P3; \text{WAIT}(t3)\]
```

The if-else activity is represented as follows in CSPM:

```
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>
</while>
```

The while activity is represented as follows in CSPM:

```
while
  <condition>
    condition
  </condition>
```
For the activity inside while, given that the corresponding QoS (response time) is \( t_i \). 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 \( t_i \), and the condition can be transformed to a conditional expression \( \text{expr} \). The CSP transformation is shown as follows:

\[
\mu P.((P;\text{WAIT}(t_i)) \triangleleft \text{expr} \triangleright ; \text{SKIP})
\]

The while activity is represented as follows in CSPM:

\[
X = \text{if expr} \quad \text{then } P;\text{WAIT}(t_i);X \quad \text{else SKIP}
\]

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:

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

For each activity \((1 \leq i \leq N)\) 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, 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, can be transformed to deterministic choices between \( P_i \). The CSP transformation is shown as follows:

\[
(P_1;\text{WAIT}(t_1)) \sqsupset
(P_2;\text{WAIT}(t_2)) \sqsupset
... \sqsupset
(P_N;\text{WAIT}(t_N))
\]

The pick activity is represented as follows in CSPM:

\[
(P_1;\text{WAIT}(t_1)) [ ]
(P_2;\text{WAIT}(t_2)) [ ]
... [ ]
(P_N;\text{WAIT}(t_N))
\]

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

```xml
<flow ...>
  <...activity1... />
</flow>
```

For each activity \((1 \leq i \leq N)\) 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 composition in Timed CSP, where each activity 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 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 shown as follows:

\[
(P_1; \text{WAIT}(t_1)) ||
(P_2; \text{WAIT}(t_2)) ||
...
(P_N; \text{WAIT}(t_N))
\]

The flow activity is represented as follows in CSPM:

\[
(P_1;\text{WAIT}(t_1))[]
(P_2;\text{WAIT}(t_2))[]
...[]
(P_N;\text{WAIT}(t_N))
\]

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 four collaborative web services: AgReq, AgRev, Al and Htl. The travelling agency uses AgReq to send service requests, and uses AgRev 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>
<thead>
<tr>
<th>Table 2. Services Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service</td>
</tr>
<tr>
<td>--------</td>
</tr>
<tr>
<td>AgReq</td>
</tr>
<tr>
<td>Al</td>
</tr>
<tr>
<td>Htl</td>
</tr>
<tr>
<td>AgRev</td>
</tr>
</tbody>
</table>

* R_t: 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.

\[
\begin{align*}
AgReqP &= AgReqPln; AgReqPOut \\
AgReqPln &= flightInfo → hotelInfo → AgReqPQoS \\
\quad \text{flightInfo} → \text{hotelInfo} → AgReqPQoS \\
AgReqPQoS &= \text{WAIT}(2) \\
AgReqPOut &= agFlight → agHotel → AgReqP \\
\quad \text{agHotel} → \text{agFlight} → AgReqP
\end{align*}
\]

Service Al is represented as the process AlP.

\[
\begin{align*}
AlP &= AlPln; AlPOut \\
AlPln &= agFlight → AlPQoS \\
AlPQoS &= \text{WAIT}(4) \\
AlPOut &= \text{flightResult} → AlP
\end{align*}
\]

Service Htl is represented as the process HtlP.

\[
\begin{align*}
HtlP &= HtlPln; HtlPOut \\
HtlPln &= agHotel → HtlPQoS \\
HtlPQoS &= \text{WAIT}(3) \\
HtlPOut &= \text{hotelResult} → HtlP
\end{align*}
\]

Service AgRcv is represented as the process AgRcvP.

\[
\begin{align*}
AgRcvP &= AgRcvPln; AgRcvPOut \\
AgRcvPln &= flightResult → hotelResult → AgRcvPQoS \\
\quad \text{flightResult} → \text{hotelResult} → AgRcvPQoS \\
AgRcvPQoS &= \text{WAIT}(1) \\
AgRcvPOut &= \text{result} → AgRcvP
\end{align*}
\]

The web service compositions AlHtl, AgReqAlHtl, and AgRcvAlHtl can be represented as process compositions as follows:

- Service composition AlHtl is a represented as the process composition AlHtl = AlP || HtlP.
- Service composition AgReqAlHtl is represented as the process composition AgReqAlHtl = AgReqP || AgReqAlHtl = \text{WAIT}(1) → AgReqP.
- Service composition AgRcvAlHtl is represented as the process composition AgRcvAlHtl = AgRcvP || AgRcvAlHtl = \text{result} → AgRcvP.

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

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

AllZero(0) 

Timed(AllZero) |
AgReqP=AgReqPln;AgReqPOut 
AgReqPln=flightInfo->flightInfo->AgReqPQoS 
flightInfo->flightInfo->AgReqPQoS 
AgReqPQoS=WAIT(2) 
AgReqPOut=agFlight->agHotel->AgReqP 
agHotel->agFlight->AgReqP 
AgRcvP=AgRcvPln;AgRcvPOut 
AgRcvPln=flightResult->hotelResult->AgRcvPQoS
```

5.3. Experimental Evaluation

Based on the CSPm 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 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 CSPm, 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.
We construct a trace $T$ according to the specification of process composition $AgReqAlHiAgRcv$ and its corresponding web service composition, $\langle flightInfo, hotelInfo, tock, tock, agFlight, agHotel, tock, tock, flightResult, tock, result \rangle$. 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 $hotelResult$ occurs; 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 $AgReqAlHiAgRcv$ in FDR based on the trace $T$, assert $AgReqAlHiAgRcv \langle has trace \rangle: \langle flightInfo, hotelInfo, tock, tock, agFlight, agHotel, tock, tock, flightResult, tock, result \rangle$. 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 $AgReq$; ($\langle HI; \rangle$; $AgRcv$) can generate specified outputs with QoS response time satisfied.

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 CSPs 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.

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.
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 CSPM assertions. In future, more details on verification of QoS-aware web service composition and more case studies will be discussed.

References


