http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 10 (6): 1194-1200, 2011
ISSN 1812-5638 / DOI: 10.3923/itj.2011.1194.1200

Modeling Web Services Composition with Timed Pi Calculus

Yuanyuan Zhang, Jun Liu, Qiong Tang and Yan Wu
College of Information Technology, Zhejiang Chinese Medical University,
Hangzhou 310053, People's Republic of China

Abstract: The Web service technology is the de facto standard to expose the functions of business applications for implementing the integration of existing e-business and improving business processes. We need to study the problem of modeling, testing and verifying the behaviors of Web services, especially in services composition. This study focuses on the issue of behavioral modeling in a service composition. It adopts the timed PI calculus to model service behaviors and interactions in a formal way. To handle the structural composition, we improve the syntax and semantic of timed PI Calculus. Based on the evolution of the timed PI calculus process, we can compose services on the fly and ensure the correctness of services composition. Our case study shows that the proposed approach has a good scalability and efficacy for service composition. In conclusion, our approach is an improvement method to modeling Web service composition.

Key words: Web services, service composition, timed PI calculus, on the fly, process evolution

INTRODUCTION

Due to the role of the World Wide Web has been changed from information interaction to service interaction, Service Oriented Computing (SOC) is an emerging paradigm for the distributed computing and e-business processing. Web services and Web service compositions (WS compositions) are currently considered to be the most widespread possibility for implementing service-oriented architectures (Papazoglou and van den Heuvel, 2006). The next wave of the internet era is being driven through the integration and composition of basic Web services available on the Web both within and across organizational boundaries. However, the building of value-added services is not a trivial task owing to the different platforms, heterogeneous programming languages and security firewalls. To truly integrate business processes across enterprise boundaries it is not sufficient to merely support simple interaction using standard messages and protocols, e.g., XML, SOAP, WSDL and UDDI specifications. These processes should provide high availability, reliability and scalability, because a failure in them can cause high economic losses, such as in B2B, B2C, G2G processes of commercial applications.

Recently, several languages for Web service composition have emerged, e.g., WSFL (http://www.ebpml.org/wsfl.htm), XLANG (http://www.ebpml.org/xlang.htm), OWL-S (http://www.w3.org/Submission/OWL-S/) and BPEL4WS (http://www.oasis-open.org/apps/org/workgroup/wsbpel/.). The goal of these languages is concentrated in defining primitives for composing services and automating service coordination in a workflow-based management way, such as the literature in Misra *et al.* (2006). Generally, based on the graph structured process models, a composite task of the Web services application is controlled and represented by a labeled directed graph in which nodes represent steps of execution and edges represent the flow of control and data among the different steps. Each step is either an atomic task or another composite task that discovered and invoked across the Internet or an enterprise intranet. However, due to the interoperability, the major fields of reliable Web service compositions research are involving services discovery, services composition, services monitoring and services maintain. Therefore, there is a need of new techniques to addressing the new requirements of the Web service environment. Among these, the time constraint is considered as an important factor in ensuring the correctness of the Web service compositions. To the best of our knowledge, none of the existing industrial standards and technological solutions can meet this need.

In most of the cases a real-time system/cyber-physical system is composed of multiple components that are working concurrently within some time constraints. The correctness of the composite software behavior depends not only on the tasks that the system is

Corresponding Author: Yuanyuan Zhang, College of Information Technology, Zhejiang Chinese Medical University, Hangzhou 310053, People's Republic of China

designed to perform but also on the physical instant at which these tasks are performed. Therefore, we are still trying to modeling, testing and verifying the behaviors of Web services.

Recently, Time Petri Nets (TPN), Timed Automata (TA) and Timed PI calculus (T_{pi}) are widely-used formalisms for the modeling and analysis of timed systems. As support tools, there exists several efficient tools like UPPAAL (Pettersson and Larsen, 2000), KRONOS (Yovine, 1997) and CMC (Laroussinie and Larsen, 1998).

Timed automata (TAs) (Alur and Dill, 1994) firstly introduced by Alur and Dill for modeling the time behaviors based on FSM, where the invariant and a clock guard are proposed as the conditions constrained in state and transition of automata, respectively. Moreover, temporal logics CTL have also been extended to deal with real-time constraints for specifying the properties, such as bounded reachability and safety. A lot of research has been proposed on the timed verification algorithms: efficient data-structures, on-the-fly algorithms, compositional methods, etc. Therefore, TAs can be used to model and reason about real-time systems such as network protocols, business processes, reactive systems, etc. And more researches of TAs has been extensively studied (Alur *et al*., 1994; Logothetis, 2005; Bouyer *et al*., 2000).

Petri Nets are a formalism developed in the '60s by C.A. Petri to model concurrent systems. In literature, time has been added to PNs in many different ways. The two main extensions of Petri Nets with time are Time Petri Nets (TPNs) and Timed Petri Nets (Saeedloei and Gupta, 2008). For these classical transition-time Petri Nets, recent work focuses check whether or not the coverability and boundedness are decidable by applying a backward (or forward) exploration technique (Abdulla and Jonsson, 2001) for timed reachability analysis. Generally, the approach for the analysis of TPNs concerns their transformation from TPNs to TAs. For example, Cassez and Roux (2008) shown how to check that a given TPN satisfies a property written in this logic. For this, they proposed a translation from TPNs to Timed Automata (TA) and check the property on the equivalent TA.

The PI calculus introduced by Milner *et al*. (1992) is a process algebra for modeling concurrency and mobility. Each component is modeled as a process. A whole system is abstracted as a set of processes among which interactions are carried out independently. The PI calculus provides a conceptual framework for describing systems whose components interact with each other. For instance, Berger (2004) considered extension of PI calculus with time and introduced (π_t-calculus) asynchronous PI calculus with timers and a notion of discrete time, locations and message failure. Lee and Zic (2002) introduced another timed extension of PI calculus called real-time PI calculus. They have introduced the time-out operator. Ciobanu and Prisacariu (2006) focusing on temporal aspects of distributed systems and introduced and studied a model called timed distributed PI calculus.

Web service exposes service functionality by interface operations descriptions. Each service composition defines that the component services interact with others by sending and receiving messages. Under time constrains, such as time-outs, the messages exchange event should be accomplished within a fixed time limit. Actually, the PI calculus is proper to model these behaviors and interactions of Web service. Thus, in this study, we present a formal approach to modeling time-related Web service composition using the technique of timed PI calculus. We model the composite Web service as timed workflow that have to respond to externally generated signals or inputs within specified time limits. Also, the future behavior of such a system depends on the time at which the external signal is received. As illustration of this methodology we use a case study which is an On-line sell system, whose description contains some time constraints. The study shows that compared with the traditional modeling technique, our approach not only improves the efficiency of Web service composition but also obtains the timed behaviors for describing the time-critical system.

MOTIVATION

A specific scenario of timed Web services composition is encountered when a customer asks a vendor agency for a business activity. Once the vendor agency receives customer request, it dynamically selects related service components, such as a login service, a Goods browse service, an express delivery service, a confirmation print service and a credit-card payment service and then composes these services into a composite service (we call it On-line sell system, OLSS). The behaviors are as follows: After login the On-line system published by the vendor, the customer choose the Goods and input the delivery address. Then the system checks the available Goods and requires the customer specifies the credit card information. Finally, the system presents a final confirmation to the customer to complete the booking. However, to accomplish the dynamic service composition with time constrains for customer is a highly complex task. The response action (or the execution action) has time constrains. This process is only valid for

a period of just half-hour which means that if any process has not been received or send in that period, the On-line system will end the business interaction.

Technically speaking, the WS composition coordinates the functionality of two Web services and is used by a third Web service. The ability to handle time is also considered a very appropriate feature because business services cannot wait forever for the reply of other parties. Timers are specialized by (1) timers can be set or started, (2) timers can be stopped and (3) timers can time-out. This can be the case when describing a particular behavior (for instance, a time-out) or stating a complex property (for example, "the alarm has to be activated within at most 10 time units after a problem has occurred").

THE TIMED PI CALCULUS

In order to model WS into the timed PI calculus process model, we first introduce the time-stamp to satisfy certain time constraints. In study of Saeedloei and Gupta (2008), messages are represented by a triple of the form <m, t_m, c>, where m is the message, t_m is the time-stamp on m and c is the clock used to generate the time-stamp. Our notion of real-time and clocks is inspired by timed automata (Alur and Dill, 1994). There are two types of clock operations (C): clock resets and constraints as follows (Saeedloei and Gupta, 2008).

$$C ::= \text{Reset} \mid \text{Constraint} \mid \epsilon$$
$$\text{Reset} ::= (\text{reset Clock}) \mid \text{Constraint (reset Clock)}$$
$$\text{Constraint} ::= ce \sim 0$$
$$ce ::= x + ce \mid x - ce \mid x$$
$$x ::= \text{Number} \mid \text{Clock}$$
$$\epsilon ::= < \mid > \mid \leq \mid \geq \mid =$$

There are two core concepts in the conventional PI calculus: processes, channels and names. The simplest entities are names. Each name has a scope and can be unbound (global) or bound to a specific process. The channels are used as links to communication by which processes interact with each other by sending and receiving messages over a channel. In this study, the channel contains three types of timed constrains: input activity $C\,\overline{a}\langle m, t_m, c\rangle$, output activity $Ca\langle m, t_m c\rangle$ and silent activity Cz. For example, $(c<6)\,\overline{a}\langle x, t_x, c\rangle$ indicates that x must be sent out on channel a within six time units since the clock c was reset.

The capabilities for actions are expressed via the prefixes of BNF, of which the syntax is defined as follows:

P::= (process)	
0 (Null process)	
| C $\overline{a}$<x, tx, c>.P	(Input process)
| Ca<x, tx, c>.P	(Output process)
| Cz.P	(Silent process)
| ζ (P+gP)	(Choice composition)
| ζ (P‖gP)	(Parallel composition)
| ζ (P; gP)	(Sequence composition)
| ζ (!gP)	(Replication composition)
| (va)P	(Restriction)
| [x = y]P	(Conditions)

Input process means that the process waits to read a value from the channel a and after having received a value u, the process continues as P but with the newly received name u replacing x, denoted as P{u/x}; Output process means that the process sends out x over the channel a and then behaves like P; Silent process means that the process can evolve to P without any actions. When an input (or output, or silent) process occurred completely, the clock c will be reset.

Here, we introduce ζ to express the global time constrains of the structural composition, where ζ is an extra formula clock $C \cap \zeta = \varnothing$. The quantifier $z \in \zeta$ is encoded for time-bounded reachability or response for Web services composition. The operators '+','‖',';'and '!' represent nondeterministic choice, parallel composition, sequence composition and replication composition, respectively. And the event g is used to describe the composition's clock invariant. For example, $(g<5)\,(P +_{<g>} P)$ indicates that the choice composition P+P must be complete within 5 time units since the clock g was reset at the start of composition.

Restriction means that the process behaves like P but the name a is local, meaning that the name cannot be used for communication with other processes; Conditions means that the process behaves as P if x and y are the same name, otherwise it does nothing.

Operational Semantics: it is used to describe the possible evolutions of a process. Sangiorgi and Walker (2001) had given as a set of transition rules focusing on input and output event. To handle the structural composition, we improve the semantics of timed PI Calculus. The operations semantic are described as follows:

Intuitively, our implementation of Table 1 is an extension of PI Calculus. We have encoded the time restriction g to each movement of process. For example, in Choice composition, the formula:

$$\frac{P \xrightarrow{C\alpha} P'}{(g \sim 0)\left(P +_{(g)} Q\right) \xrightarrow{C\alpha} P'}$$

means that if per-condition $P \xrightarrow{C\alpha} P'$ is satisfied then the choice composition process should be accomplished

Table 1: Semantics of timed PI calculus

Behaviors of web service	Operational semantics
Choice composition	$CHO := \frac{P \xrightarrow{C\alpha} P'}{(g \sim 0)\ (P +_{<g>} Q) \xrightarrow{C\alpha} P'}$
	$CHO - RES := \frac{reset\ c, P \xrightarrow{(reset\ c)C\alpha} P}{(g \sim 0)(reset\ c)(P +_{<g>} Q) \xrightarrow{C\alpha} P}$ where $\alpha \in \{\tau, a(x, t_x, c), \bar{a} < y, t_y, c >\}$
Parallel composition	$PAP := \frac{P \xrightarrow{C\alpha} P'}{(g \sim 0)\ P \|_g Q \xrightarrow{C\alpha} P' \| Q}$,
	$PAP - RES := \frac{reset\ c, P \xrightarrow{(reset\ c)C\alpha} P'}{(g \sim 0)(reset\ c) P \|_g Q \xrightarrow{(reset\ c)C\alpha} P' \| Q}$ where $bn(\alpha) \cap fn(Q) = \varnothing$;
Sequence composition	$SEQ := \frac{P \xrightarrow{a<x, t_x, c>} P', Q \xrightarrow{\bar{a}<x, t_x, c>} Q'}{(g \sim 0)\ P;_g Q \xrightarrow{\tau} P'; Q'}$,
	$SEQ - RES := \frac{reset\ c, P \xrightarrow{(reset\ c)\, a<x, t_x, c>} P', Q \xrightarrow{(reset\ c)\, \bar{a}<x, t_x, c>} Q'}{(g \sim 0)\ (reset\ c)\ P;_g Q \xrightarrow{\tau} P'; Q'}$
Restriction	
	$CLOSE := \frac{P \xrightarrow{a<x, t_x, c>} P', Q \xrightarrow{\bar{a}<x, t_x, c>} Q'}{P; Q \xrightarrow{\tau} (vx)\ P' \| Q'}$ if $x \notin fn(Q)$,
	$CLOSE - RES := \frac{reset\ c, P \xrightarrow{(reset\ c)a<x, t_x, c>} P', Q \xrightarrow{(reset\ c)\bar{a}<x, t_x, c>} Q'}{(g \sim 0)(reset\ c)\ P;_g Q \xrightarrow{\tau} (vx)\ P' \| Q'}$, f $x \notin$ fn (Q)
Replication composition	$REP := \frac{P \xrightarrow{(g \sim 0)a<x, t_x, c>} P'}{(g \sim 0)!_g !P \xrightarrow{(g \sim 0)a<x, t_x, c>} P' \| !P}$,
	$REP - RES := \frac{reset\ c, P \xrightarrow{(reset\ c)a<x, t_x, c>} P'}{(g \sim 0)\ (reset\ c)!_g P \xrightarrow{(reset\ c)a<x, t_x, c>} P' \| !P}$

within the g time units which described in time restriction g over C. The process also can be formalized with a reset command (resect c) for clean all clock variants during the choice composition, namely, the formula is denoted as:

$$\frac{(reset\ C\ )P \xrightarrow{(reset\ C\ )C\alpha} P'}{(g \sim 0)(reset\ C\ )\left(P +_{\langle g \rangle} Q\right) \xrightarrow{(reset\ C\ )C\alpha} P'}$$

Similarly, in Parallel composition, when bn (α) $\cap$fn (Q) = $\varnothing$ which indicates that action α is not compatible with the process Q, the parallel composition $\|$ is asynchronous. In Sequence composition, one service output messages Ca<m, t_m, c> which can be received as input message $C\bar{a}$ <m, t_m, c> by the other service. Note that the sequence composition introduces the silent action during services composition. In Restriction composition, suppose x can't be received by the invoked service Q that x$\notin$fn (Q), the restriction operation (va) P is used to monitoring the restriction. In replication composition, it is required to be acted with the time restriction g for implementing the $(!P \xrightarrow{(g \sim 0)a<x, t_x, c>} P' \| !P)$ loop operations.

IMPLEMENTING THE TIME CALCULUS FOR WS COMPOSITION

We first give out our architecture to reflect the success of service execution and interactions. As Fig. 1 shown, in the service layer, each Web service has time element that involves the service execution time, the starting time for execution, the service binding time and timed input/output descriptions. In the services candidate layer, services pool is used to provide the alternative service when current service fails. In workflow model, the Web service-based software system in general is modeled as a workflow under control processes.

In previous study, the BPEL4WS is used specify the invocation of WS composition and the WSDL is used to expose the functionality of Web services. This section presents methods of the translation from BPEL4WS specification of WS composition to Timed PI calculus.

For service layer, we can model the input (output, or silent) process for the each WS according WSDL (Web Service Definition Language). In the industrial standard service description language WSDL, there are four types of operations, i.e., one-way, request-response, solicit-response and notification. In our study, WSDL is improved that <time stamp="">, <clock name=""> and <clockOperations/> are introduced to describe the timed behaviors of the service interface. The corresponding PI calculus are translated as follows:

Algorithm 1: One-Way description

```
<operation name="a">
  <input message="x" >          ➜ (c<9) ā< x, t_x, c>
       <time stamp ="tx" />
  <clock name="c">
  <clockOperations>
       <clock expression="c<9">
</clockOperations>
</input >
</operation>
```

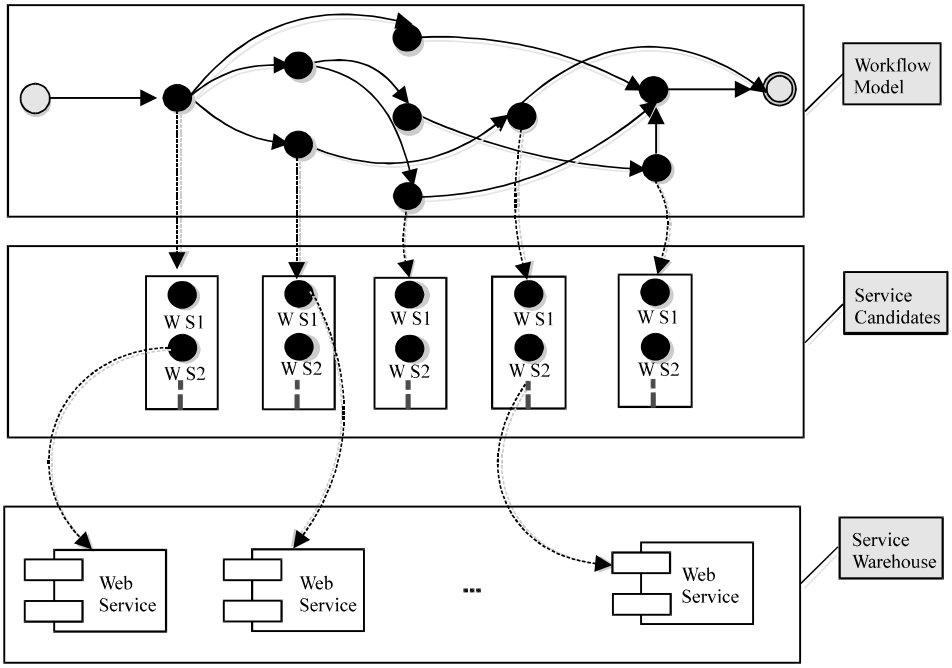

Fig. 1: The architecture of a timed modeling framework

Algorithm 2: Request-Response description

```
0 <operation name="a">
<input message="x" >          ➜(c<9) ā<x,tx,c>.(d<3) a<y, ty, c>
<time stamp ="tx" />
<clock name="c">
<clockOperations>
<clock expression="c<9">
</clockOperations>
</input >
<output message="y" >
<time stamp ="ty" />
<clock name="d">
<clockOperations>
<clock expression="d<3">
</clockOperations>
</out>
</operation>
```

Algorithm 3: Solicit-Response description

```
0<operation name="a">
<output message="y" >         ➜(d<3)a<y,ty,c>.(c<9) ā<x, tx, c>
<time stamp ="ty" />
<clock name="d">
<clockOperations>
<clock expression="d<3">
</clockOperations>
</out>
<input message="x" >
<time stamp ="tx" />
<clock name="c">
<clockOperations>
<clock expression="c<9">
</clockOperations>
</input >
</operation>
```

Algorithm 4: Notification description

```
<operation name="a">
  <output message="x" >       ➜ (c<9)a<x, tx, c>
        <time stamp ="tx" />
  <clock name="c">
  <clockOperations>
        <clock expression="c<9">
</clockOperations>
</output >
</operation>
```

Algorithm 1-4 show that each type of operation can be modeled as a process expression with the additional time constrains. Notice that, an operation of the request-response type or solicit-response type can be represented by the combination of a one-way type operation and a notification type operation.

For work flow layer, the BPEL describe the specification of the composite WS where the < receive> and < reply > activities are used to describe the interface constraints of WS including time constraints. The <invoke> activities are used to represent interaction to other services. It can be mapped to a sequential composition that invoke send input parameters and obtains the message from the invoked service. In additional, there are two kinds of <invoke> activities in BEPL, namely, synchronous and asynchronous. They can be mapped to the parallel and sequence composition. The while activity introduces an iteration control and requires a loop structure to represent the repetition. The pick activity combines a switch activity applied to various sequences of other activities. In our practice, we extend the BEPL to broaden their applicability to the timed structural behaviors, such as the tag <time constrains>.

Algorithm 5: A sequence description

```
<sequence>
<time constrains="g">          ➜ (c<3) Pa; (g)Pb
  <clockOperations>
       <clock expression="g<3">
</clockOperations>
  <clock reset="g"/>
  <receive operation ="a" >
  <invoke operaction="LoginService">
       <reply operation ="a" >
  <receive operation ="b" >
  <invoke operaction="BookingHotel">
       <reply operation ="b" >
       </time >
</sequence >
```

For example, in Algorithm 5, we depict how the sequence composition is formulated where a global g is added. It means that at the beginning the global clock is reset and the process of sequence execution needs to be invoked within 3 time units.

CASE STUDY

Consider again the example shown in section II. In this section, we will discuss how the business logic model of the campsite WS (OLSS) is built with the help of timed PI calculus. The service behavior refers to the dynamic properties of a service which includes the actions the service can take, the states the service has and the message exchange sequence supported by the service. In Fig. 2, the interactions of OLSS are specified via six states (P_{login}, P_{view}, $P_{delivery}$, $P_{creditcard}$, P_{print}, P_{end}) and seven channels (C1, C2, C3, C4, C5, C6, C7). In this design, it is assumed that services receive the message through their corresponding channel. C1, C2, C7 are notification-type which output a messages succ, fail, mess with different time constrains, respectively. C3, C4, C5, C6 are the one way-type operations which get input messages selectgood, onLinePay and cashPay, respectively.

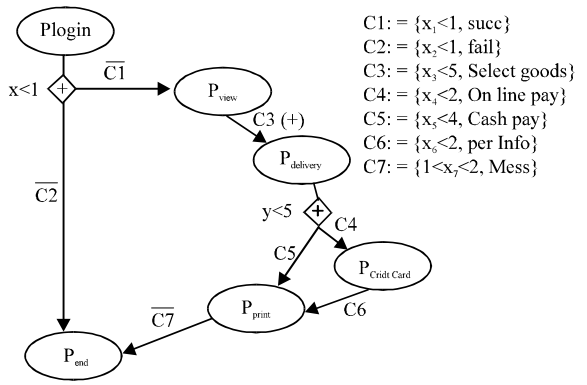

Fig. 2: A composite service

Using timed PI calculus to model the behavior of a service, we can define the whole service as a timed PI calculus process.

P:== (reset(x)) (x<1) ((x_2<1) $\overline{C2}$ <succ, t_{succ}, x_2>+$_x$ ((x_1<1) $\overline{C1}$ <fail, t_{fail}, x_1>.(x_3<5) $\overline{C3}$ <select Good, $t_{selectGood}$, x_3>. ((reset (y)) (y<1) ((x_5<4) $\overline{C5}$ <cashPay, $t_{cashPay}$, x_5>+$_y$ ((x_4<2) $\overline{C4}$ <onLinePay, $t_{onLinePay}$, x_4>.(x_6<2) $\overline{C6}$ <perInfo, $t_{perInfo}$, x_6>)).(1<x_7<2) $\overline{C7}$ <mess, t_{mess}, x_7>)))

According to the operational semantics, we simulate the process of each step of business logic. In Fig. 2, there are three message exchange sequences which can eventually come to an end (denoted as 0). From the operational semantics of the exchange sequences, we can build a composite service correctly on the fly.

The first message exchange sequence is gotten as follows:

P $\xrightarrow{(reset\ c)(x<1)}$ ((x_2<1) $\overline{C2}$ <succ, t_{succ}, x_2>+$_x$ ((x_1<1) $\overline{C1}$ <fail, t_{fail}, x_1>.(x_3<5) $\overline{C3}$ <select good, $t_{selectGood}$, x_3>. ((reset (y)) (y<1) ((x_5<4) $\overline{C5}$ <cashPay, $t_{cashPay}$, x_5>+$_y$ ((x_4<2) $\overline{C4}$ <onLinePay, $t_{onLinePay}$, x_4>. (x_6<2) $\overline{C6}$ <perInfo, $t_{perInfo}$, x_6>)). (1<x_7<2) $\overline{C7}$ <*mess*, t_{mess}, x_7>))) $\xrightarrow{(x_2<1)<s_{ucc}, t_{succ}, x_2>}$ 0.

The second message exchange sequence is gotten as follows:

P $\xrightarrow{(reset\ c)(x<1)}$ ((x_2<1) $\overline{C2}$ <succ, t_{succ}, x_2>+$_x$((x_1<1) $\overline{C1}$ <fail, t_{fail}, x_1>. (x_3<5) $\overline{C3}$ <select good, $t_{selectGood}$, x_3>. ((reset (y)) (y<1) ((x_5<4) $\overline{C5}$ <cashPay, $t_{cashPay}$, x_5>+$_y$ ((x_4<2) $\overline{C4}$ <onLinePay, $t_{onLinePay}$, x_4>. (x_6<2) $\overline{C6}$ <perInfo, $t_{perInfo}$, x_6>)). (1<x <2) $\overline{C7}$ <mess, t , x_7>))). $\xrightarrow{(x_1<1)<fail, t_{fail}, x_1>}$ (x_3<5) $\overline{C3}$ <select good, $t_{selectGood}$, x_3>. ((reset (y)) (y<1) ((x_5<4) $\overline{C5}$ <cashPay, $t_{cashPay}$, x_5>+$_y$ ((x_4<2) $\overline{C4}$ <onLinePay, $t_{onLinePay}$, x_4>. (x_6<2) $\overline{C6}$ <perInfo, $t_{perInfo}$, x_6>)). (1<x_7<2) $\overline{C7}$ <mess, t_{mess}, x_7>)).

$\xrightarrow{(x_3<5)<selectGood, t_{selectGood}>}$ \((reset (y)) (y<1) ((x_5<4) $\overline{C5}$ <cashPay, $t_{cashPay}$, x_5>+$_y$ ((x_4<2) $\overline{C4}$ <onLinePay, $t_{onLinePay}$, x_4>.(x_6<2) $\overline{C6}$ <perInfo, $t_{perInfo}$, x_6>)). (1<x_7<2) $\overline{C7}$ <mess, t_{mess}, x_7>)).

$\xrightarrow{(reset\ (y))(y<1)}$ ((x_5<4) $\overline{C5}$ <cashPay, $t_{cashPay}$, x_5>+$_y$ ((x_4<2) $\overline{C4}$ <onLinePay, $t_{onLinePay}$, x_4>.(x_6<2) $\overline{C6}$ <perInfo, $t_{perInfo}$, x_6>)).(1<x_7<2) $\overline{C7}$ <mess, t_{mess}, x_7>)

$\xrightarrow{(x_4<1)<cashpay, t_{cashpay}, x_5>}$ 0.

The second message exchange sequence is gotten as follows:

$P \xrightarrow{(\text{restes c})\,(x,1)} ((x_2<1)\ \overline{C2}\ \langle \text{succ}, t_{succ}, x_2\rangle +_x ((x_1<1)\ \overline{C1}\ \langle \text{fail}, t_{fail}, x_1\rangle.(x_3<5)\ \overline{C3}\ \langle \text{select good}, t_{selectGood}, x_3\rangle.((\text{reset }(y))\ (y<1)\ ((x_5<4)\ \overline{C5}\ \langle \text{cashPay}, t_{cashPay}, x_5\rangle +_y ((x_4<2)\ \overline{C4}\ \langle \text{onLinePay}, t_{onLinePay}, x_4\rangle.(x_6<2)\ \overline{C6}\ \langle \text{perInfo}, t_{perInfo}, x_6\rangle)).(1<x_7<2)\ \overline{C7}\ \langle \text{mess}, t_{mess}, x_7\rangle)))$

$\xrightarrow{(x_1<1)\langle fail,\ t_{fail},\ x_1\rangle} (x_3<5)\ \overline{C3}\ \langle \text{select good}, t_{selectGood}, x_3\rangle.((\text{reset }(y))\ (y<1)\ ((x_5<4)\ \overline{C5}\ \langle \text{cashPay}, t_{cashPay}, x_5\rangle +_y ((x_4<2)\ \overline{C4}\ \langle \text{onLinePay}, t_{onLinePay}, x_4\rangle.(x_6<2)\ \overline{C6}\ \langle \text{perInfo}, t_{perInfo}, x_6\rangle)).(1<x_7<2)\ \overline{C7}\ \langle \text{mess}, t_{mess}, x_7\rangle))$

$\xrightarrow{(x_1<1)\langle selectsGood,\ t_{selectgood},\ x_3} ((\text{reset }(y))\ (y<1)\ ((x_5<4)\ \overline{C5}\ \langle \text{cashPay}, t_{cashPay}, x_5\rangle +_y ((x_4<2)\ \overline{C4}\ \langle \text{onLinePay}, t_{onLinePay}, x_4\rangle.(x_6<2)\ \overline{C6}\ \langle \text{perInfo}, t_{perInfo}, x_6\rangle)).(1<x_7<2)\ \overline{C7}\ \langle \text{mess}, t_{mess}, x_7\rangle))$

$\xrightarrow{(\text{restes }(y))\ (y<1)} ((x_5<4)\ \overline{C5}\ \langle \text{cashPay}, t_{cashPay}, x_5\rangle +_y ((x_4<2)\ \overline{C4}\ \langle \text{onLinePay}, t_{onLinePay}, x_4\rangle.(x_6<2)\ \overline{C6}\ \langle \text{perInfo}, t_{perInfo}, x_6\rangle)).(1<x_7<2)\ \overline{C7}\ \langle \text{mess}, t_{mess}, x_7\rangle)$

$\xrightarrow{(x_4<2)\langle onLinePay,\ t_{onLinePay},\ x_4\rangle} (x_6<2)\ \overline{C6}\ \langle \text{perInfo}, t_{perInfo}, x_6\rangle.(1<x_7<2)\ \overline{C7}\ \langle \text{mess}, t_{mess}, x_7\rangle$

$\xrightarrow{(x_6<2)\langle perInfo,\ t_{perInfo},\ x_6\rangle} (1<x_7<2)\ \overline{C7}\ \langle \text{mess}, t_{mess}, x_7\rangle$

$\xrightarrow{(1<x_7<2)\langle mess,\ t_{mess},\ x_7\rangle} 0.$

CONCLUSION AND FUTURE WORK

In the complex, distributed, open, dynamic and unpredictable environments, how to ensure the correctness of Web Services composition plays a critical role in SOA. It is becoming well-admitted that the use of formal methods is hot top in software engineering, such as Petri Nets, Automata and PI calculus. In this study, we introduce timed PI calculus processes to describe the services and their interactions. In the particular case study we have used to illustrate how this methodology works. Present future work will focus on automatic checking so as to implement a tool supporting this verification. Many existing tools can help us to do the verification automatically, such as the tool MWB. So, we plan to extend the MWB to check the timed process.

ACKNOWLEDGMENT

The authors would like to thank the editors and anonymous referees for their suggestions and the remarkable improvements they brought to this paper. This paper has been supported by the research fund of the Zhejiang Chinese Medical University (2010ZY14).

REFERENCES

Abdulla, P.A. and B. Jonsson, 2001. Ensuring completeness of symbolic verification methods for infinite-state systems. Theor. Comput. Sci., 256: 145-167.

Alur, R. and D.L. Dill, 1994. A theory of timed automata. J. Theor. Comput. Sci., 126: 183-235.

Alur, R., L. Fix and T.A. Henzinger, 1994. A determinizable class of timed automata. Proceedings of the 6th International Conference on Computer Aided Verification, (CAV'98), Springer-Verlag, London, UK., pp: 1-13.

Berger, M., 2004. Towards abstractions for distributed systems. Technical Report.

Bouyer, P., C. Dufourd, E. Fleury and A. Petit, 2000. Are timed automata updatable. Proceedings of the 12th International Conference Computer Aided Verification (CAV), July 2000, Springer, Chicago, IL, USA., pp: 464-479.

Cassez, F. and O.H. Roux, 2008. From Time Petri Nets to Timed Automata. In: Petri Net, Theory and Applications, Kordic, V. (Ed.). I-Tech Education and Publishing, Vienna, Austria.

Ciobanu, G. and C. Prisacariu, 2006. Timers for distributed systems. Electron. Notes Theor. Comput. Sci., 164: 81-99.

Laroussinie, F. and K.G. Larsen, 1998. CMC: A tool for compositional model-checking of real-time systems. Proceedings of the IFIP Joint International Conference on Formal Description Techniques and Protocol Specification, Testing and Verification, (FORTE-PSTV'98) Kluwer Academic Publishers, Dordrecht, pp: 439-456.

Lee, J.Y. and J. Zic, 2002. On modeling real-time mobile processes. Aust. Comput. Sci. Commun., 24: 139-147.

Logothetis, G., 2005. Forward symbolic model checking for real time systems. Proc. Asia South Pacific Design Autom. Conf., 2: 1043-1046.

Milner, R., J. Parrow and D. Walker, 1992. A calculus of mobile processes, I. Inform. Comput., 100: 1-40.

Misra, R.B., S. Srinivasan and D.P. Mital, 2006. The use of web services technology in the design of complex software interfaces: An educational perspective. Inform. Technol. J., 5: 1127-1131.

Papazoglou, M.P. and W.J. van den Heuvel, 2006. Service-oriented design and development methodology. Int. J. Web Eng. Technol., 2: 412-442.

Pettersson, P. and K.G. Larsen, 2000. UPPAAL2k. Bull. Eur. Assoc. Theor. Comput. Sci., 70: 40-44.

Saeedloei, N. and G. Gupta, 2008. Timed PI calculus. http://www.utdallas.edu/~gupta/tpi.pdf.

Sangiorgi, D. and D. Walker, 2001. PI Calculus: A Theory of Mobile Processes. Cambridge University Press, New York, USA..

Yovine, S., 1997. KRONOS: A verification tool for real-time systems. Int. J. Software Tools Technol. Transfer, 1: 123-133.