Partial Order Reduction for Verification of WS-BPEL

Denduang Pradubsuwun and Wutthipong Kongburan

***Abstract*—Web service-business process execution Language (WS-BPEL) is a promising language for describing a web service composition. Unfortunately, WS-BPEL lacks expressive power of formal semantics to support verifiability. Consequently, timed trace theory has been used to assure the correctness and reliability of the web service composition with timing constraints. Moreover, since most of business processes are more complicated, the cost of verification is high. It is likely that the state explosion problem will take place. In order to tackle this problem, in this paper a partial order reduction algorithm for timed trace theoretic verification has been applied to minimize the number of states. Experimenting with a tourism information system, the proposed approach expresses its effectiveness.**

***Index Terms*—Partial order reduction, timed trace theory, web service composition.**

I. Introduction

Web service composition is an ability to integrate existing web services together to fulfill the practical requirements. Presently, WS-BPEL (Web Service–Business Process Execution Language) [1] has been introduced to composite web services. Since creating composite web services is more complex task, formal verification of WS-BPEL is essential to guarantee the correctness and reliability of web service compositions with timing constraints [2]. However, WS-BPEL has some defect on verification.

Recently, a number of research efforts on the verification of the web service composition have been activated including [3]-[8]. Even though all of them are successful in unraveling weaknesses of verification of web service composition, they do not consider timing constraints of business processes. Therefore, [9]-[13] have been proposed formal semantics for verifying WS-BPEL with timing constraints. Whereas they are not able to detect safety and timing failures, timed trace theoretic verification proposed in [14] does. Using this approach, a specification and implementations are modeled by time Petri net [15] and conformance relation between them is checked.

However, since practically most of businesses processes not only involve functionality but also involve the timing constraints, they are quite complicated interactions. The cost of verification is rather high. It is likely that the number of states may explode or the state explosion problem will occur. Hence, a partial order reduction algorithm for timed trace theoretic verification [16] has been applied to reduce the number of states while verifying the web service composition with timing constraints.

This paper is organized as follows. Timed trace theoretic verification and its partial order reduction algorithm are described in the Section II. How to transform WS-BPEL into time Petri net is explained in the Section III. Then, experimenting with a case study is demonstrated in the Section IV. Finally, the conclusion is given in the Section V.

Manuscript received January 31, 2014; revised April 2, 2014.

The authors are with the Department of Computer Science, Thammasat University, Pathumthani, Thailand (e-mail: denduang@cs.tu.ac.th; oatt_fanclub@hotmail.com).

II. Partial Order Reduction for Timed Trace Theoretic Verification

Since our approach relies on the framework of partial order reduction for timed trace theoretic verification proposed in [16], we briefly its idea in this section.

A. Timed Trace Theoretic Verification

We introduce the important notions of timed trace theory including module, semimodule and time trace structure.

A *module* M is a tuple (*I, O, N*), where *I* am a set of input transition, *O* is a set of output transition, and *N* is a time Petri net. A time Petri net is a six-tuple of Petri net $N = (P, T, B, F, M_0, Time)$

1) $P = \{p_1, p_2, \ldots, p_m\}$ is a finite nonempty set of *places*.
2) $T = \{t_1, t_2, \ldots, t_n\}$ is a finite set of *transitions* ($P \cap T = \emptyset$).
3) B is the backward incidence function $P \times T \to N$ or *input arc* flow relation from place to transition.
4) F is the forward incidence function $T \times P \to N$ or *output arc* flow relation from transition to place.
5) $M_0 \subseteq P$ is *initial marking* of the time Petri net.
6) *Time* is delay function Let Q^+ denote the set of positive rational numbers $T \to Q^+ \times Q^+ \cup \{\infty\}$, $T = [a_i, b_i]$ such that a_i, b_i represent the static bound for the *earliest* and *latest firing times* of transitions, respectively, satisfying $a_i(t) \le b_i(t)$ for all $t \in T$.

A *semimodule* is the same as a module except the definition of its timed trace structure.

A *timed trace structure* of a module M, denoted by T(M), is a tuple (*I, O, S, F*) where *S* is called success trace set and *F* is called failure trace set, contains a trace $y(w, \tau) \notin S$ shown as (1).

- $y \in F$, or
- $y \in S$, $\tau \le \mathrm{TL}(y, N)$, $w \in I$, or
- $y \in S$, $\tau > \mathrm{TL}(y, N)$, limit $(y, N) \subseteq I$. (1)

TL(y, N) is the latest time until when the firing of all enabled transitions in N can be postponed after y. limit (y, N) is the set of wires that correspond to the enable transitions which determine TL(y, N).

Equation (2) must be considered in order to check the correctness between a module M_1 and a module M_2. We use notion $T_1 = (I_1, O_1, S_1, F_1)$ and $T_2 = (I_2, O_2, S_2, F_2)$ such that I_1

DOI: 10.7763/IJMO.2014.V4.380

$\cup O_1 = I_2 \cup O_2$. Intersection of T_1 and T_2, denoted by $T_1 \cap T_2$, is shown as follows.

$$(I_1 \cap I_2, O_1 \cup O_2, S_1 \cap S_2, (P_1 \cap F_2) \cup (F_1 \cap P_2)) \quad (2)$$

If $(P_1 \cap F_2) \cup (F_1 \cap P_2) = \emptyset$ then the module M_1 *conform to* the module M_2. It means that the module M_1 behaves similarly to the module M_2 with any environment. From this definition, the following theorem is inherited.

Theorem 1: $\{M_1,\ldots, M_{k\text{-}1}, M_{K1},\ldots, M_{km}, M_{k+1},\ldots, M_n\}$ conforms to M_s, if $\{M_{k1},\ldots, M_{km}\}$ conforms to M_k, and $\{M_1,\ldots, M_{k\text{-}1}, M_k, M_{k+1}, \ldots, M_n\}$ conforms to M_s

Note that $M_1,\ldots, M_{k\text{-}1}, M_{K1},\ldots, M_{km}, M_{k+1},\ldots$, and M_n are represented as implementation modules and M_s is represented as a specification module of a system.

Refer to the verification algorithm in [16] which is derived from Theorem1; *safety failure* will occur when an output produced by a (semi)module cannot be accepted by some other (semi)module. On the other hand, *timing failure* will happen when an input expected by a (semi)module cannot be given in time by some other (semi)module. Both types of failure are shown in Fig. 1 and Fig. 2 respectively.

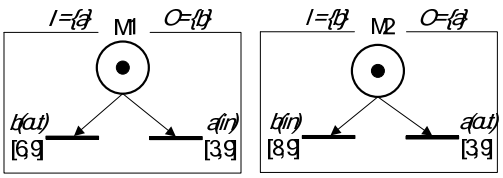


Fig. 1. Module that cause a safety failure.

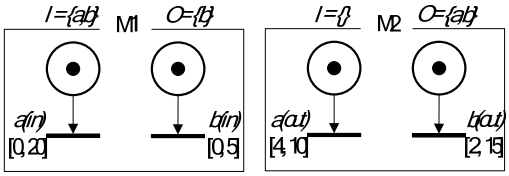


Fig. 2. Module that cause a timing failure.

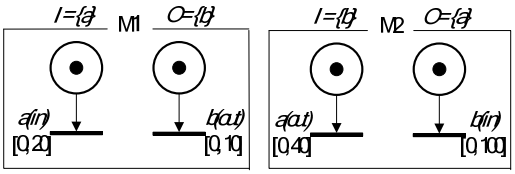
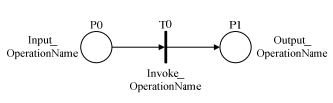
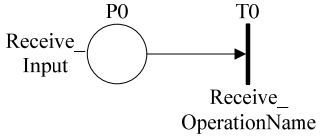
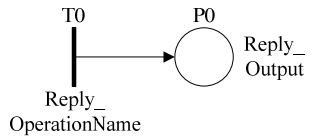
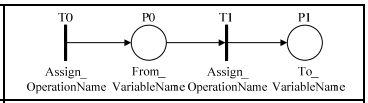
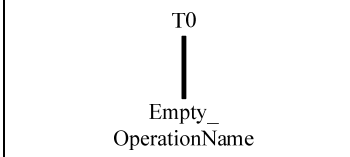
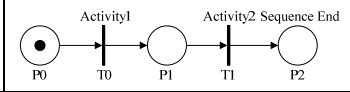
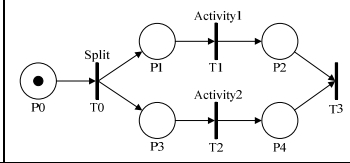
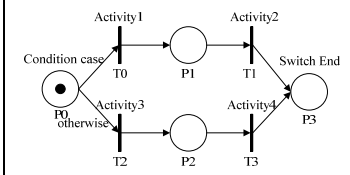
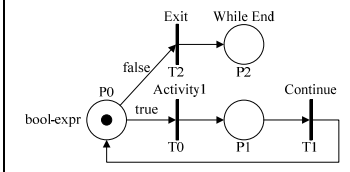
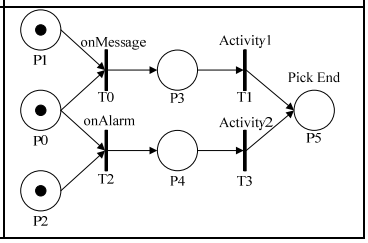


Fig. 3. Transition *a* (out) hides a timing failure.

TABLE I: MAPPING FROM WS-BPEL ACTIVITY TO TIME PETRI NET

Basic activity	Time Petri net
<invoke partnerLink=" Seller" portType="SP:Purchasing" operation="Purchase" inputVariable="sendPO" outputVariable="receivePO" /invoke>	
<receive partnerLink=" NCName" portType="QName" operation="NCName" variable="BPELVariableName" /receive>	
<reply partnerLink=" NCName" portType="QName" operation="NCName" variable="BPELVariableName" /reply>	

<assign copy from variable="var1" to variable="var2" /assign>	
<empty do nothing /empty>	
<sequence> activity 1 activity 2 </sequence>	
<flow> activity 1 activity 2 </flow>	
<switch> <case condition cond1> activity1 activity2 </case> <otherwise> activity3 activity4 </otherwise> </switch>	
<while> <condition> bool-expr </condition> activity1 </while>	
<pick> <onMessage standard-attributes> activity1 </onMessage> <onAlarm standard-attributes> (<for> duration-expr </for> | <until> deadline-expr </until>) activity2 </onAlarm> </pick>	

B. Partial Order Reduction for Timed Trace Theoretic Verification

A partial order reduction algorithm is used to avoid a complete enumeration set of the state space. Some subset of possible states will be produced as long as the correctness is not affected. We call a state space obtained by the partial order reduction algorithm *the reduced state space*. The partial order reduction algorithm for timed trace theoretic verification has been proposed in [16]. The reduced state space must satisfy the three rules including [PT1], [PT2] and [PT3]. [PT1] is necessary because a new deadlock state must not be generated in the reduced state space. [PT2] is for handling conflict transitions. When a transition that conflicts with another one is fired, the conflicting transition or its ancestor should also be fired. [PT3] is for handling transitions hiding timing failures. To understand [PT3], we firstly define a firable transition. It is an enabled transition that can fire earlier than any other enabled transitions. [PT3] defines that limiting transition which is a firable output transitions such that its latest firing time point is smallest among the firable output transitions of all modules. For example, consider the modules illustrated in Fig. 3. A set of output firable transition at the current state consists of *a* (out) and *b*(out). If *b* (out) fires firstly, then a timing failure occurs because *a* (out) can fires later than *a* (in). On the other hand,

if *a* (out) fires before firing a transition *b* (out), a timing failure is never detected because *a* (out) hides the timing failure. Thus, [PT3] forces *b* (out) to fire, if *a* (out) is firstly chosen to fire.

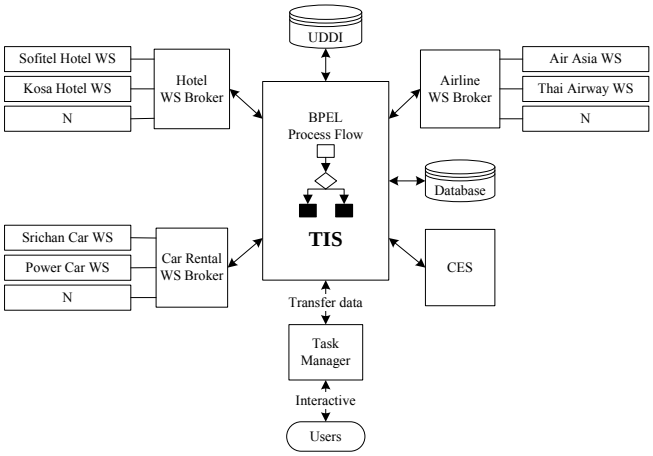


Fig. 4. Architecture of a tourism information system.

III. How to Transform WS-BPEL into Time Petri Net

In order to verify web service compositions, the WS-BPEL describing a web service composition must be transformed into time Petri nets by our algorithm which is proposed in [14]. The brief idea of the algorithm is shown as follows.

1) A state of web service is represented by the place.
2) A web service activity is represented by the transition.
3) A message or a variable is represented by the token.
4) A control flow relation between activities is represented by the firing rule and the arc.
5) A timing constraint of the web service activity is mapped to delay function.

In this work, we only consider the most frequently used workflow pattern in WS-BPEL. There are two types of activities including the basic and structure activity. The basic activity includes invoke, receive, reply, assign, and empty activity. The structure activity includes sequence, flow, switch, while, and pick activity. Time Petri net derived from each activity of WS-BPEL is relatively straightforward. Mapping basic and structure activity is illustrated in Table I.

A. Basic Activity

1) The <invoke> activity enables us to call an operation which offered by service. The operation can be asynchronous one-way or synchronous request-response. An asynchronous invoke requires only the input variable of operation. Synchronous invoke requires both an input variable and an output variable.
2) The <receive> activity specifies the received information, the port type and operation form the partner link to invoke. This activity waits for an asynchronous callback response message from a service.
3) The <reply> activity allows the process to send a message to a previous request. It must happened after a receive activity.
4) The <assign> activity provides a method for data manipulation, such as copying the contents of one variable to another one.
5) The <empty> activity enables us to insert a no-operation instruction into a process.

B. Structure Activity

1) The <sequence> activity is a container where activities are performed sequentially following the order in the sequence element. The activity is complete when the last activity in the sequence is finished.
2) The <flow> activity is a container where enclosed activities concurrently execute. Activities within a flow start simultaneously and the flow finished when the activities are complete.
3) The <switch> activity consists of an ordered list of one or more conditional branches defined in a case statement. The branches are considered following the order in the switch element. The switch activity is finished when the activity of the selected statement completes.
4) The <while> activity executes a contained activity repeatedly as long as its condition evaluates to true. Otherwise, the contained activity do not execute at all.
5) The <pick> activity waits for the occurrence of exactly one event from a set of events, and then executes the activity associated with that event. After one of events is selected, the pick will no longer accept other events.

C. Timing Constraints

To make business process more efficient, timing constraints should be included in a process model. Thus, we need model to represent WS-BPEL with timing constraints. In this paper, we define them by using the *annotation* concept [11]. Timing constraint annotations are defined separately from a WS-BPEL. In some scenarios, the execution time of an activity is often nondeterministic, but may be within bounds. Therefore, a time interval can be used to represent a flexible execution time of activity. Note that the time interval of an activity can be obtained from domain experts or derived from the history log. The time interval based approach allows us to model duration time of each activity. A simple WS-BPEL with a timing constraint annotation is shown as below. For example, an activity “T3” is one of basic activities and has the interval 4.10 units of time. It means that the activity T3 must be executed from 4 until 10 units of time.

```
<process name = “CompositeProcess”>
<partnerLinks> … </partnerLink>
<variables> … </variable>
<sequence>
<invoke name= “Tinquire”>
<sequence>
<flow>
  <sequence>
    <invoke name= “T2”>
    <invoke name= “T3”>
  </sequence>
</flow>
</sequence>
<invoke name= “Tchoose”>
<switch>
```

```
  <case condition= “bpws:getVariableData('S')
=0”>
    <invoke name= “T6”/>
  </case>
  <case condition= “bpws:getVariableData('S')
=1”>
    <invoke name= “T7”/>
  </case>
</switch>
</sequence>
</process>

<?The timing constraint annotation layer of the
process “process” ?>
<process : CompositeProcess rdf :ID = “T”>
< process: AtomicProcess rdf : resource = “#T2” />
  Time(T3) = 4..10
< process: AtomicProcess rdf : resource = “#T3” />
  Time(T4) = 4..10
< process: AtomicProcess rdf : resource = “#T6” />
  Time(T6) = 6..12
< process: AtomicProcess rdf : resource = “#T7” />
  Time(T6) = 6..10
< process: AtomicProcess rdf : resource =
“#Tinquire” />
  Time(T6) = 1..5
< process: AtomicProcess rdf : resource =
“#Tchoose” />
  Time(T6) = 5..6
< /process: CompositeProcess >
```

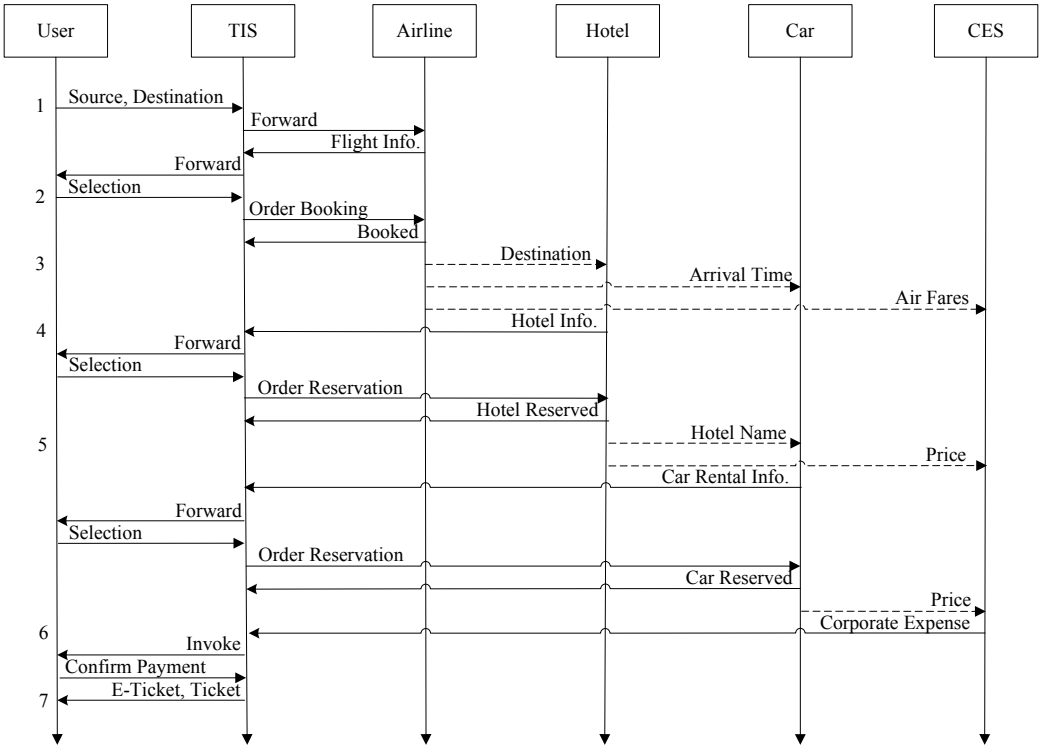


Fig. 5. Sequence diagram of a tourism information system.

IV. The Case Study

In this section, the partial order reduction algorithm for timed trace theoretic verification has been applied to minimize the number of state while verifying the web service composition with timing constraints. We demonstrated with a *Tourism* Information *System* (TIS) [17]. This system is a flight ticket, hotel, and car rental reservations. The TIS architecture is illustrated in Fig. 4. The TIS is a coordination middleware or central control engine. It consists of three broker web services including *Airline Broker Web Service, Hotel Broker Web Service,* and *Car Rental Broker Web Service.* All broker web services perform as a representative so that they will call services from business providers within their groups and send results to the TIS. In this experiment, a main scenario shown in Fig. 5 is verified.

TIS are a coordination middleware or central control engine. It consists of three broker web services including *Airline Broker Web Service, Hotel Broker Web Service,* and *Car Rental Broker Web Service.* They perform as a representative so that they will call services from business providers within their groups and send results to the TIS.

A. The Specification of the Tourism Information System

The business process starts receiving initial requirements from customers through *task manager* service. Then the TIS sends information to be processed by all broker web services including *Airline Broker Web Service, Hotel Broker Web Service,* and *Car Rental Broker Web Service.* Customers may reserves a flight ticket and hotel, but probably not a car rental. Lastly, corporate *expense service* (CES) calculates the expense for the customer.

Airline Broker Web Service is a business process service agents used to book a seat. The agent includes partner *Thai Airway* web service and *AirAsia* web service. These services show a list of flight and wait for a customer requirement

Hotel Broker Web Service is a business process service agents used for hotel reservations. The agent includes *Sofitel*

hotel web service and *Kosa* hotel web service. These services show hotel information and wait for a customer requirement.

Car Rental Broker Web Service is a business process service agents used for renting a car. The agent includes partner Srichan car rental web service and *Power* car rental web service. These services show car rental information and wait for user requirement. The specification of the TIS is modeled in Fig. 6.

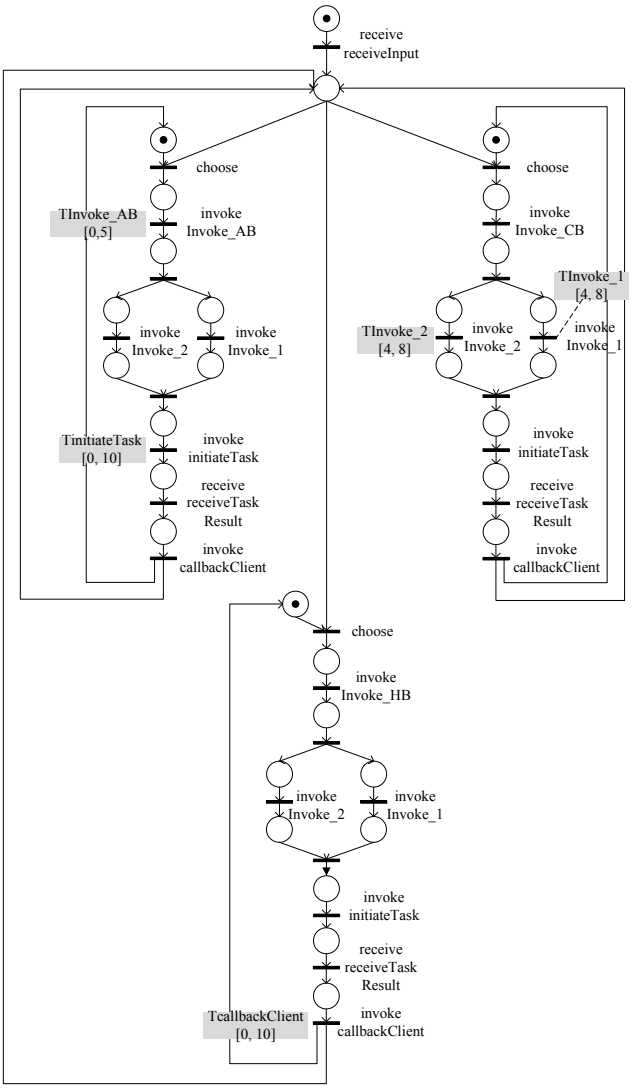


Fig. 6. Time petri net of a specification of tourism information system.

The TIS must satisfy the following properties.

1) After the TIS get request from a customer, the service will forward some information to the airline broker web service within 5 units of time.
2) The airline broker web service will search and display flight information corresponding to a customer request. It must respond to the customer within 10 units of time.
3) The hotel broker web service will accept a customer request, such as find, select and reserve a room, and respond to the customer within 10 units of time.
4) The car rental broker web service will accept a customer request and forward it to partners simultaneously from 4 until 8 units of time.

B. The implementation of the Tourism Information System

The WS-BPEL implementation of TIS is obtained from [17]. And, it is transformed into time Petri net by using the algorithm proposed in [14]. Time Petri net implementation of TIS, *Airline Broker Web Service*, *Hotel Broker Web Service*, and Car *Rental Broker Web Service* are shown in Fig. 7, Fig. 8, Fig. 9 and Fig. 10, respectively.

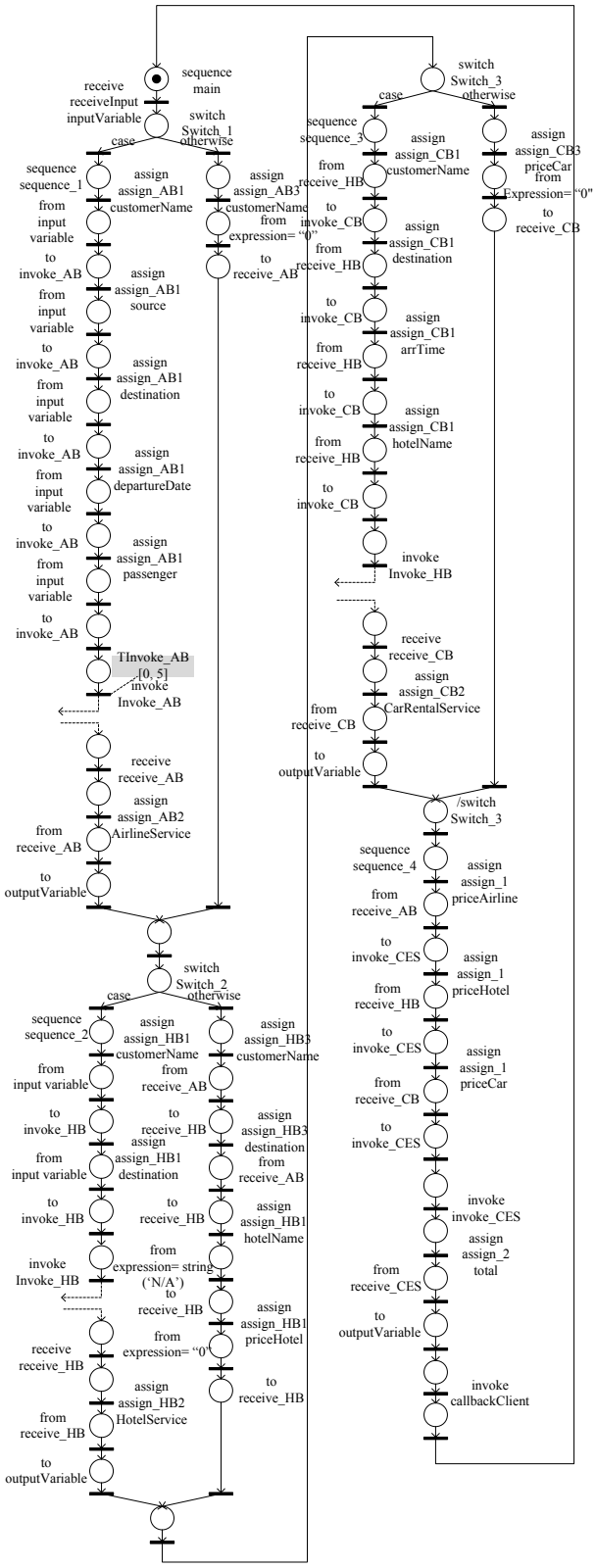


Fig. 7. Time Petri net of the implementation of tourism information system.

C. Verification Result of the Tourism Information System

The experiments have been done on a 2.10 GHz Intel core i7 with 4 gigabytes of memory. We have verified the TIS by

tool in [16]. We then give some result obtained with the analysis of our nontrivial case study. The CPU times and memory usage for verification of the TIS are 0.04 sec. and 271 kilobytes, respectively.

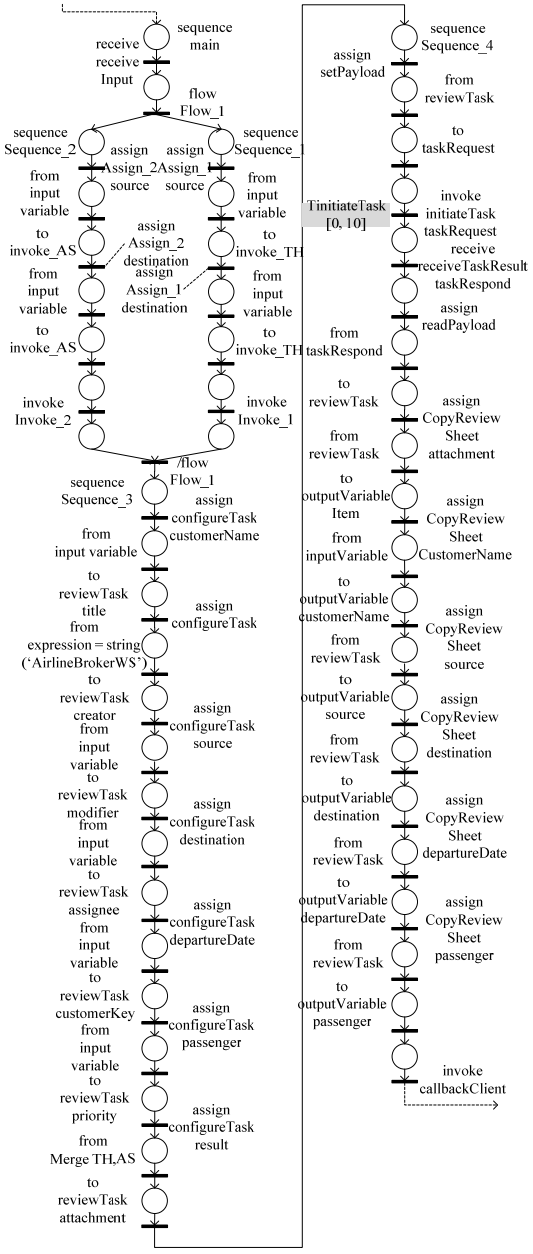
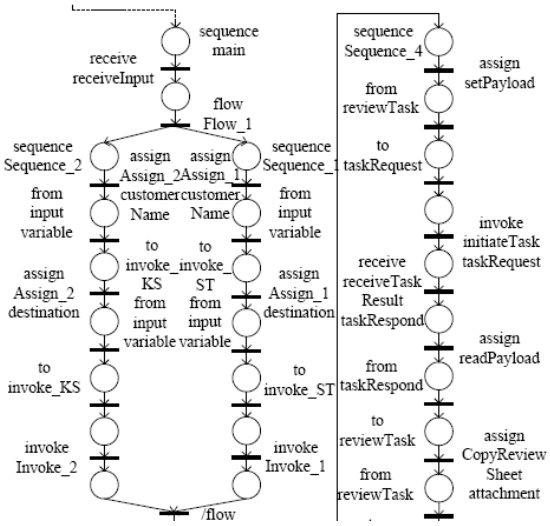


Fig. 8. Time Petri net of the implementation of airline broker web service.



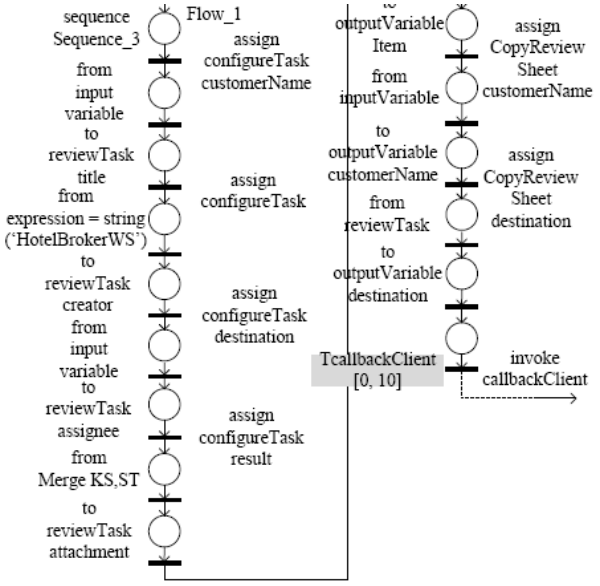


Fig. 9. Time Petri net of the implementation of hotel broker web service.

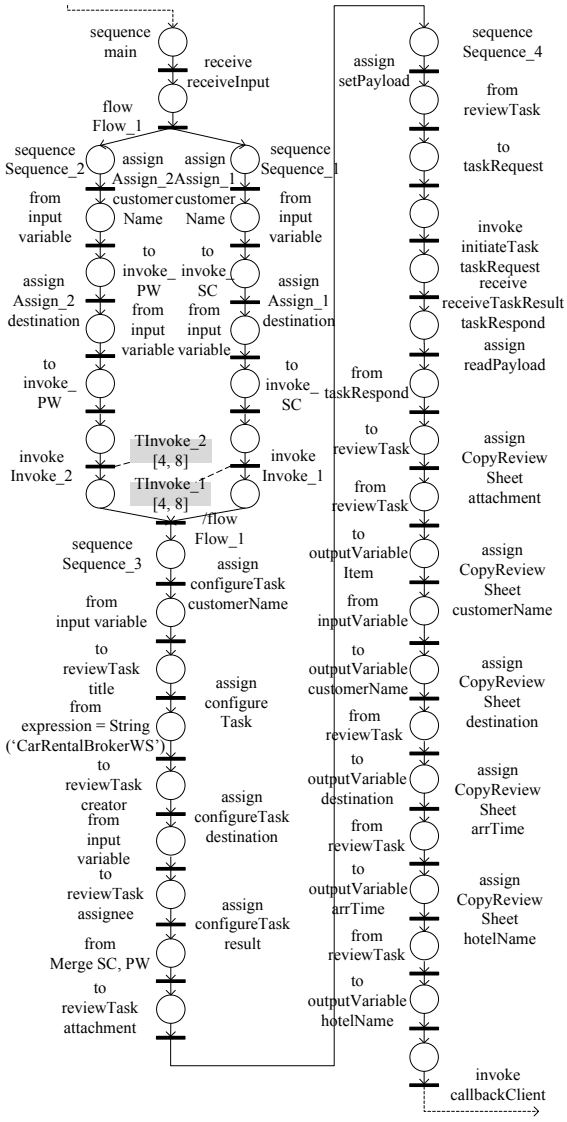


Fig. 10. Time Petri net of the implementation ofcar rental broker web service.

V. Conclusion

In this paper, we have applied a partial order reduction algorithm for timed trace theoretic verification to assure the

correctness and reliability of web service composition with timing constraint. We demonstrated the effectiveness of our proposed method to verify a tourism information system. In the future, we are planning to verify a practical system by using the hierarchical verification method. According to our framework, we are also implementing a tool in order to automatically verify the whole process of web service composition.

REFERENCES

[1] A. Alves *et al*. (April 2007). Web Service Business Process Execution Language Version 2.0. [Online]. Available: http://docs.oasis-open.org/wsbpel/ 2.0/wsbpel-v2.0.pdf

[2] N. Guermouche and S. D. Zilio, "Towards timed requirement verification for service choreographies," presented at 8th International Conference on Collaborative Computing: Networking, Applications and Worksharing, Pittsburgh, PA, United States, October 14-17, 2012.

[3] S. Hinz, K. Schmidt, and C. Stahl, "Transforming BPEL to Petri nets," in *Proc. the Third International Conference on Business Process Management (BPM 2005)*, France, pp. 220-235, 2005.

[4] N. Lohmann. (August 2007). A Feature-Complete Petri Net Semantics for WS-BPEL 2.0 and its Compiler BPEL2oWFN. Available: http://www.informatik.uni-rostock.de/tpp/publications/Lohmann_2007_hubtr212.pdf.

[5] Y. Yang, Q. Tan, J. Yu, and F. Liu, "Transformation BPEL to CP-Nets for verifying web services composition," in *Proc. the International Conference on Next Generation Web Services Practices*, Washington, DC, USA, pp.137-142, 2005.

[6] Y. Yang, Q. Tan, and Y. Xiao, "Verifying web services composition based on hierarchical colored Petri nets," presented at First International Workshop on Interoperability of Heterogeneous Information, Bremen, Germany, October 31 - November 5, 2005.

[7] H. Kang, X. Yang, and S. Yuan, "Modeling and verification of web services composition based on CPN," presented at IFIP International Conference on Network and Parallel Computing, Dalian, China, September 18-21, 2007.

[8] Y. Wang and S. Pan, "CPN-Based verification of web service composition model," presented at International Conference on Educational and Information Technology. Chongqing, China, September 17-19, 2010.

[9] E. Martinez, M. E. Cambronero, G. Diaz, and V. Valero, "Design and verification of web services compositions," presented at 2009 Fourth International Conference on Internet and Web Applications and Services, Venice/Mestre, Italy, May 24-28, 2009.

[10] J. Mei, H. Miao, Q. Xu, and P. Liu, "Modeling and verifying web service applications with time constraints," presented at 9th IEEE/ACIS International Conference on Computer and Information Science, Kaminoyama (Yamagata), Japan, August 18-20, 2010.

[11] G. Dai, R. Liu, C. Zhao, and C. Hu, "Timing constraints specification and verification for web service compositions," presented at 2008 IEEE Asia-Pacific Services Computing Conference, Yilan, Taiwan, December 9-12, 2008.

[12] R. Liu, C. Hu, C. Zhao, and Z. Gao, "Verification for time consistency of web service flow," presented at Seventh IEEE/ACIS International Conference on Computer and Information Science, Portland, Oregon, USA, May 14-16, 2008.

[13] W. Song, X. Ma, C. Ye, W. Dou, and J. Lu, "Timed modeling and verification of BPEL processes using time Petri nets," presented at 2009 Ninth International Conference on Quality Software, Jeju, South Korea, August 24-25, 2009.

[14] W. Kongburan and D. Pradubsuwun, "Formal verification of WS-BPEL using timed trace theory," presented at 5th KKU International Engineering Conference, Khonkaen, Thailand, March 27-28, 2014.

[15] P. Bonhomme, G. Berthelot, P. Aygalinc, and S. Calvez, "Verification technique for time Petri nets," presented at 2004 IEEE International Conference on System, Man and Cybernetics, The Hague, Netherlands, October 10-13, 2004.

[16] D. Pradubsuwun, T. Yoneda, and C. Myers, "Partial order reduction for detecting safety and timing failures of timed circuits. *IEICE Trans. 88-D)*, pp. 1646-1661, July 2005.

[17] E. Naenudorn, "One stop service for tourism business using web service composition model," M.S. thesis, Dept. Information Technology, Khonkaen University, Khonkaen, Thailand, 2006.

Denduang Pradubsuwun received the B.S. degree with the 2nd class honors in computer science from Ramkhamhaeng University, Bangkok, Thailand, in 1995, the M.S. degree in computer science from Chulalongkorn University, Bangkok, Thailand in 1999, and the D.Eng. degree in computer science from the Tokyo Institute of Technology, Tokyo, Japan in 2005.

He is a lecturer in the Department of Computer Science, Faculty of Science and Technology, Thammasat University, Pathumthani, Thailand. His current research interests are formal verification, timed circuits verification, and concurrent system.

Wutthipong Kongburan received the B.S. degree in computer science from Thammasat University, Pathumthani, Thailand, in 2010.

Currently, he is a computer technical officer at Ramkhamhaeng University, Bangkok, Thailand. His research interests include software verification, formal method and software engineering.