2010 Symposia and Workshops on Ubiquitous, Autonomic and Trusted Computing

Towards Trustworthy Composite Service Through Business Process Model Verification

Wenjia Huai, Xudong Liu, Hailong Sun
School of Computer Science and Engineering, Beihang University
Beijing, China
{huaiwj, liuxd, sunhl}@act.buaa.edu.cn

***Abstract*—The Business Process Modeling Notation (BPMN) is a standard for modeling business processes in the early phases of systems development. Model verification is an important means to guarantee the trustiness of composite services. The verification of model, especially the model with strict time constraints, is a challenge in the field of trust composite services. Whether a model is trustworthy depends not only on the model structure but also on quantitative properties such as time properties. In this paper, we propose a mapping from BPMN to time Petri nets, and use the verification techniques on the basis. The algorithm we present can be used to check the model structure and the time choreography.**

Keywords-trustworthy composite service, BPMN, model verification, time Petri nets, time property

I. INTRODUCTION

With the rapid development of Internet, the Service Oriented Computing (SOC) builds the software systems through choreographing services based on Service Oriented Architecture (SOA), which becomes a new concept. Currently, SOC has been widely used, the process modeling languages have arose in a historic moment. The Business Process Modeling Notation (BPMN) was developed by Business Process Management Initiative (BPMI). The primary goal of BPMN is to support business process management for both business users and technical users by providing a notation that is intuitive yet able to represent complex process semantics[13].

A model is the base to driven the software development. However, with the increasing of demands and design complexity, model faults will occur inevitably. The model errors contain not only the model structure faults such as deadlocks, but also the quantitative properties choreography errors such as time properties. Before the model has been executed, discover and correct the mistakes as early as possible, in order to guarantee the model is trustworthy. It has the potential for avoiding considerable loss in time, money and human resources. We use BPMN to modeling composite services, shown in Fig.1. How to provide a unified approach to verify the BPMN model is one of the important research contents.

This paper presents a method for BPMN model verification based on time Petri nets. The method supports the analysis of model structure, and can test the time properties of the model. First, we translate the BPMN model to the time Petri net. Secondly, construct the reachability graph of the Petri net and verify the model structure. Furthermore, give the time choreography verification algorithm to test the time conflicts. In order to catch all the possible timed conflicts, we give the clock constraint to the tasks and message flows. The clock constraint aims at making explicit the implicit timed conflicts when services are interacting together.

To summarize, in this paper we make the following contributions: (1) we propose a unified method for BPMN model verification based on time Petri nets. In the premise of ensuring the execution semantics, we map BPMN models to time Petri nets directly. (2) The verification of BPMN models contains the structure and the time properties. The structure analysis of the model is more comprehensive, including dead tasks, deadlocks and infinite loops. Unlike existing time analysis, we consider the expected time property of services before the execution, and detect the implicit time choreography conflicts by the analysis.

The paper is organized as follows. Section 2, we review the basic concepts of BPMN and time Petri nets. Section 3 depicts the mapping rules how to transform the BPMN to Petri nets. Section 4 shows how to checking the static structure problems of the model. Furthermore, we give the method to check the time conflicts. Section 5 reports the prototype systems implementation and the case study. Section 6 discusses the related works. Some conclusions are given in the last section.

II. PRELIMINARIES

A. BPMN

The BPMN process is consistent of BPMN elements and has the ability to depict the process involving multiple services. This paper focuses on the executable elements of BPMN. It does not deal with the non-functional features (i.e., pools, lanes) which do not affect the execution.

978-0-7695-4272-0/10 $26.00 © 2010 IEEE
DOI 10.1109/UIC-ATC.2010.114

BPMN process models are mainly composed of flow objects and connecting objects.

Flow objects are the core graphical elements to define the behavior of a business process and consist of events, activities and gateways. An event is something that occurs during the course of a business process. Events consist of start events, intermediate events, and end events. An activity usually stands for work to be performed. A gateway is used to control the divergence and convergence of a process. Connecting objects, including sequence flows and message flows, are responsible for linking flow objects and make up composite services processes. Sequence flows show the executive order of the process in a pool. The different processes in separated pools want to exchange messages must use message flows.

Case Scenario: RDOA (Request Day Off Approval)

Let us present a simple request day off approval case that we use to show the related issues of the proposed approach. Below, we give some time requirements.

Get the result of the preliminary approval requires at least 56 hours and at most 72 hours. Leadership verify services may require the participation of a company's leadership and need at least 60 hours. Return the result at least 12 hours after getting the result of verify. Once the staff submit the request for leave, he must receive the reply within 120 hours and then arrange the follow-up works.

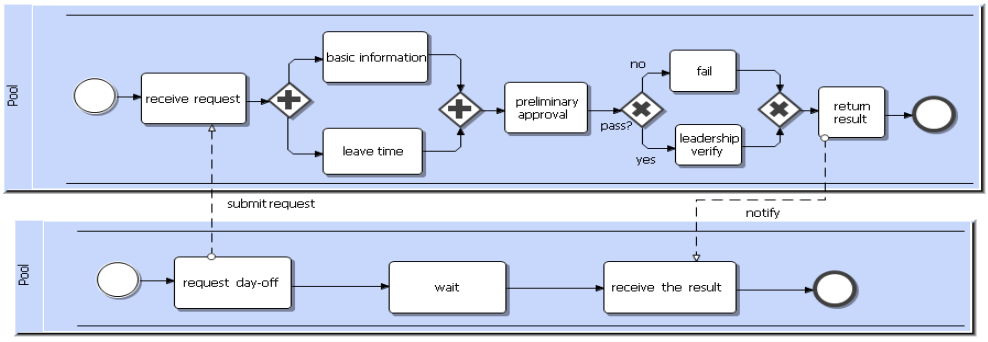


Figure 1. Request day off approval process

B. *Time Petri Nets*

Definition 1. A time Petri net is a five-tuple, N= {P, T, F, L, M_0}, where

- P $\{p_1, p_2, \ldots, p_n\}$ is a finite set of places;
- T $\{t_1, t_2, \ldots, t_n\}$ is a finite set of transitions;
- F $\subseteq$ (P$\times$T)$\cup$(T$\times$P) is a set of directed arcs;
- L [eft, lft] are the earliest and latest time constraints of places, and have three forms of expressions: [eft, lft] (eft$\leqslant$lft, lft$\neq\infty$), [eft,∞] and [0,lft] (lft$\neq\infty$);
- M_0 is the initial marking. And $\bullet\Gamma$ denote the pre-set and the notation$\Gamma\bullet$ denote the post-set.

Definition 2. A Petri net is a workflow net (WF-net), if and only if [2]:

- PN has two special places: i and o. Place i is a source place : $\bullet$ i= $\emptyset$. Place o is a sink place: o $\bullet$ =$\emptyset$.
- If we add a transition t_k to PN which connects place o with i (i.e. $\bullet t_k$ ={o} and $t_k \bullet$ ={i}), then the resulting Petri net is strongly connected.

In this paper, we aim at generating a time Petri net that conforms to a workflow net. Because a workflow net can depict the executed status of a model intuitively. Markings in the workflow net are used to characterize its running states. Thus we can further analyze the static and dynamic behaviour of composite services. Another reason is model verification requires a definite start state and a definite end state. Composite services with multiple processes may have multiple start and end states. The behaviour of such process is not clear, leading to difficult verification. Intuitively, the initial state of a workflow net is a single token located in the source place, and the correct end state is a single token reached the end place. Therefore we can test the model of composite service conveniently.

III. MAPPING BPMN ONTO TIME PETRI NETS

A. *Mapping Flow Objects*

Fig.2 shows the mapping from BPMN flow objects, which contain events, tasks and gateways to Petri net modules. In accordance with the execution semantics, a gateway is classified into concurrency and choice. Concurrency is divided into the fork gateway with a single input and multiple outputs, and the join gateway with multiple inputs and a single output. Similarly, choice is classified into the decision gateway and merge gateway. Gateways are mapped onto Petri net modules with transitions capturing their routing behaviours. For fork gateways, we model the concurrency forking conditions in the outgoing flows as places that have a common transition as input. Thus, when the transition is fired, every place will get a token. We do not model the conditions

themselves, but the translated Petri net module has the execution semantics.

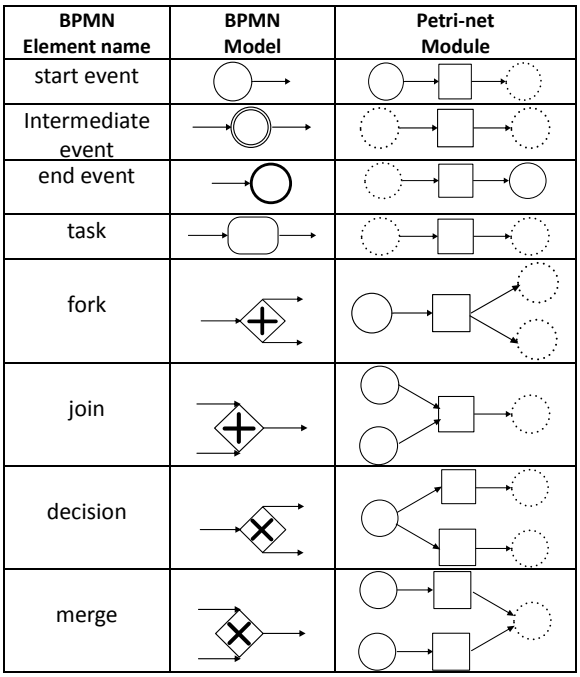
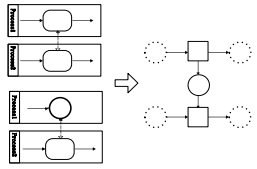


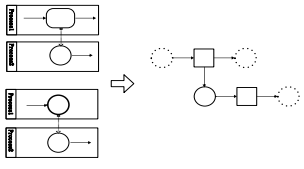
Figure 2. Mapping flow objects onto Petri net models

B. Mapping Connecting Objects

Sequence flows stands for the executed order in BPMN process and don't have semantics. So we map them to arcs in Petri net directly. Message flows may be with the attribute named MessgaeRef which identifies the interaction between processes. In order to guarantee the semantics it can be represent as a place with an incoming arc coming from a transition stands for a send action and an outgoing arc points to a transition stands for a receive action. Especially, the message flow links to a start event. It can map to an arc link from the transition of the send action to the place of the start event directly, shown in Fig.3 (b).



(a) to task



(b) to start event

Figure 3. Mapping connecting objects onto Petri net models

C. Mapping to Time Petri Nets

The time property of a task means the time constraints need to be satisfied when the task completed. We translate it to the time constraint for the task's place changing from 1 to 0. Because message flows are mapping to places. The interaction of messages between different processes may need to meet certain time constraints. The time property should add to not only the tasks in a process, but also the delivery of messages between different processes. This method has access to the actual conditions. Therefore we get the equivalence time Petri net models. Then, add the source state and end state for the time Petri net model, so as to satisfy the requirements of the workflow net.

IV. VERIFYING BPMN MODELS WITH TIME PROPERTIES

A. Model Structure Verification

The reachability graph of Petri nets reflects the dynamic characteristic and it covers all reachable states of the model, can describe the execution (delivery) of tasks (or message flows) definitely, and can be used to check errors of the logical structure. A reachability graph G =<V, E> of Petri net is a directed graph. The set of vertices (V) is the states of Petri net, where $M_i \in V$, that is $V=\{ M_i (p_0 , p_1 , \cdots , p_n)\}$, $p_j = \begin{cases} 0 ; \text{doesn't hold tokens} \\ 1; \text{ hold one token} \end{cases} (0 \le j \le n)$. The set of directed edges (E) represents a directed arc from one state to another state. Building the reachability graph with the algorithm in [5], then verify the model structure on the basis. We can check the following properties:

- dead task: A task can never be executed for any case.
- deadlock: A case gets stuck in some state where it is not possible to execute any tasks[4].
- loop: A case is trapped in a loop where it is possible to execute tasks but no real progress is possible[4]. Let δ be a path in G of the form $\delta=M_0 \rightarrow M_1 \rightarrow \ldots \rightarrow M_n$. If all M_i are distinct and there are $M_k (i<k<n)$ such that $M_k=M_i$, then we say that the subsequence $\delta_i = M_i \rightarrow M_s \rightarrow \ldots \rightarrow M_k$ ($M_k=M_i$) is a loop in G.

First verify dead tasks in Petri net model. The edges include all transitions which have been fired, if a transition exists in the model but doesn't exist in Edges, its corresponding task is a dead task. Next, test the deadlock. Find the leaf nodes in G, that is, the outdegree of a node is zero. If it has $M_i \in$Leaf, $M_i(p_{end} =0)$, and p_{end} is the end place, so M_i is the marking of deadlock. Finally, we test the loop, starting from the initial state M_0, if the node has not been processed, we recursive call the depth first search for traversing the reachability graph. If a new node has been found in the prefix path, copy it to the loop path. If not, copy it to the current path. Traverse all the executable

paths in the reachability graph, in order to verify the loop path.

B. Time Choreography Verification

Let δ be a Petri net executable path, and δ_i is an arbitrary prefixes of δ. For a reasonable path which meets the time constraints, its prefix δ_i is necessary to satisfy the corresponding time constraints. Test the prefix path, if it has time conflicts, there are time conflicts in the path. If all the executable paths in the reachability graph have no time conflicts, the time choreography of the model is reasonable.

For an executable path, the time choreography verification algorithm is depicted as follows. Initialize the clock constraint for places with 0, from the initial state (2-3 lines), observe the change of states caused by transition, and compare the change of token in places. When fire the executable transition t_i , the state changes from M_x to M_y. According to the changes of token,

$$\text{token} \begin{cases} 1 \to 0 \text{ The task just completed;} \\ 0 \to 1 \text{ The task has just been activated;} \\ 1 \to 1 \text{ The task remains active;} \\ 0 \to 0 \text{ The task has been completed or has not been activated yet} \end{cases}, \text{ if}$$

the token of p_i changes from 1 to 0, and the token of p_j changes from 0 to 1, that means when the transition t_i has been fired, p_i has completed, while p_j has just been activated. That goes to show p_j occurs after p_i, therefore the clock constraint c_i bound to be accumulated to c_j. At this moment, p_i task completed, its time property $[eft_i, lft_i]$ also need to add to c_j. If the token of place maintains 0 or 1, it indicates the current fired transition has no effect for the place. We don't treat this situation. If there is a latest time property lft_i for p_i, and p_i has the upper limit clock constraint clt_i , then the time of the activated p_j meet the cumulative value, the earlist time property is also treated like this (6-8 lines) . If p_i doesn't have the latest time, that is lft_i is infinite, so only deal with the earlist time (9 line). One task is completed, several tasks are activated, and the activated tasks should compute the time, respectively. Multiple tasks completed, one task is activated, the activated task needs to meet multiple time constraints, time conflicts may occur (10-11 lines). If do not have time conflicts, continue to deal with the path until the end state, that shows the path is correct (12-15 lines).

Algorithm 1. Time Choreography Verification

Input: Path= $<M_0 \xrightarrow{t_0} M_1 \xrightarrow{t_1} \dots \xrightarrow{t_i} M_{end}>$

Output: Boolean

1. TimeChoreography
2. { current_path= (M_0 , t_0 , M_0')
3. c[place_number] = 0;
4. **repeat** current_path =(M_x , t_i , M_y) ;
5. for each p_i {1 → 0} **&&** p_j {0 → 1}
6. { **if** (lft_i ! = ∅)
7. { cet_j = cet_i +eft_i ;
8. clt_j = clt_i +lft_i ; }
9. **else** cet_j = cet_i +eft_i ; }
10. **if** ($cet_j \le c_j \le clt_j$ **&&** $cet_j' \le c_j \le clt_j'$ **&&** ($cet_j' > clt_j$ || $cet_j > clt_j'$))
11. **return** false;
12. **else** { cet_j = compareMax (cet_j, cet_j') ;
13. clt_j = compareMin (clt_j, clt_j') ; }
14. **until** $M_y == M_{end}$;
15. **return** true; }

V. System Implementation and Case Study

A. Prototype System Design and Implementation

Fig.4 shows the system structure. A BPMN file serves as input to the prototype system. After model transformation, structure verification and time choreography analysis, we get conclusions. If the model is correct, we may deploy it to runtime environments. If not, give the location of the fault markings or paths.

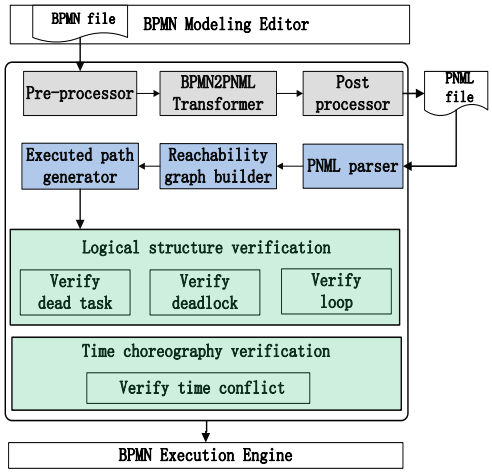


Figure 4. Prototype system structure

We use the BPMN modeling editor as a graphical editor to create BPMN models. The prototype takes a BPMN file as input, then the Pre-processor implemented preprocess, such as adjusting the structure of the BPMN file or numbering specific elements of the BPMN. BPMN2PNML transformer consists of a set of mapping rules which specify how to transform the elements of BPMN to Petri net modules. These mapping rules are implemented by XSLT code. After the post process, exports the resulting Petri net in the form of a PNML file.

PNML Parser loads the PNML file and the Reachability Graph Builder constructs the reachability graph of Petri nets, and the Executed path generator computes executed paths on the basis. The structure verification contains the analysis of dead tasks, deadlocks and loops. Time choreograph verification module verify the time conflict of the model. The verification module is implemented by Java.

B. *Case Study: Request Day Off Approval*

Using the mapping methods, we translate the case RDOA shown in Fig.1 into the time Petri net model shown in Fig.5.

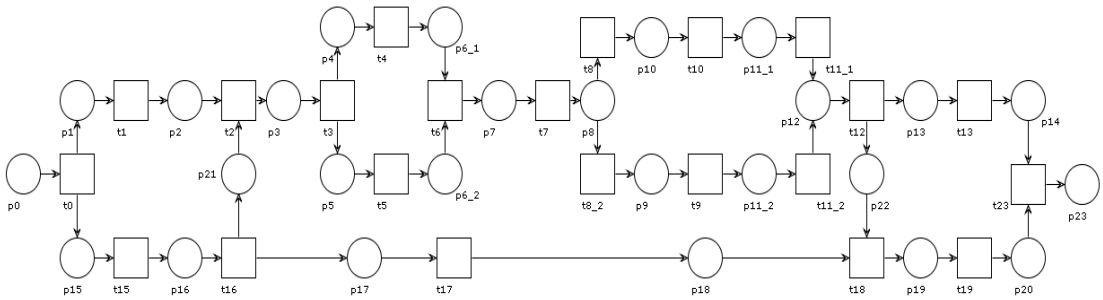


Figure 5. RADO time Petri net model

P_0 and t_0 are the start place and transition of the workflow net. P_{23} and t_{23} are the end place and transition. After getting the time Petri net model, we verify the structure of the model. We found the reachability graph has a single leaf node $M_{41}\{p_{23} = 1\}$, and p_{23} is the end place, so the model doesn't exist deadlocks. Moreover, the instance doesn't have loops. So the structure of this model is correct. Next, we analyze the time properties. We choose one path to demonstrate the process of testing. $M_0 \xrightarrow{t_0} M_1 \xrightarrow{t_{15}} M_3 \xrightarrow{t_1} M_4 \xrightarrow{t_{16}} M_6 \xrightarrow{t_{17}} M_9 \xrightarrow{t_2} M_{11} \xrightarrow{t_3} M_{14} \xrightarrow{t_5} M_{17} \xrightarrow{t_4} M_{19} \xrightarrow{t_6} M_{21} \xrightarrow{t_7} M_{24} \xrightarrow{t_{8\_2}} M_{28} \xrightarrow{t_9} M_{31} \xrightarrow{t_{11\_2}} M_{33} \xrightarrow{t_{12}} M_{35} \xrightarrow{t_{18}} M_{37} \xrightarrow{t_{19}} M_{39} \xrightarrow{t_{13}} M_{40} \xrightarrow{t_{23}} M_{41}$. The marking of the reachability graph involved are shown below.

	p_2	p_4	p_5	p_9	p_{13}	p_{14}	p_{10}	p_7	p_{12}	p_8	p_{11_1}	p_{11_2}	p_3	p_{6_1}	p_{6_2}	p_{16}	p_{17}	p_{18}	p_{19}	p_{20}	p_{21}	p_{22}	p_0	p_1	p_{15}	p_{23}
M_0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
M_1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0
M_3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0
M_4	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
M_6	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0
M_9	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0
M_{11}	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0
M_{14}	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
M_{17}	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0
M_{19}	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0
M_{21}	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
M_{24}	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
M_{28}	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
M_{31}	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0
M_{33}	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
M_{35}	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0
M_{37}	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

Figure 6. RADO reachability graph marking

Begin from the initial state M_0, compare the states from M_0 to M_1, find $p_0\{1 \to 0\}$, meanwhile p_{15},p_{23} $\{0 \to 1\}$, c_0=0, l_0=0, and figure out c_{15}[0,0] , c_{23}[0,0]. Compare the next states from M_1 to M_3, p_5 $\{1 \to 0\}$, p_{16} $\{0 \to 1\}$, and according to the previous step we know c_{15}[0,0], and figure out c_{16}[0,0]. According to the algorithm, we treat along the path followed until the step $M_{21} \xrightarrow{t7} M_{24}$, appears time constraint, now p_7 $\{1 \to 0\}$, p_8 $\{0 \to 1\}$, so we get the relationship p_8 occurs after p_7. In the case l_7[56,72] has been know already, so c_8[56,72]. The state from M_{24} to M_{28}, p_8 $\{1 \to 0\}$, $p_9\{0 \to 1\}$, so c_9[56,72]. Next, $M_{28} \xrightarrow{t9} M_{31}$, p_9 $\{1 \to 0\}$, $p_{11\_2}\{0 \to 1\}$, we get the relationship p_{11_2} occurs after p_9, owing to l_9[60,∞], and the previous step c_9 [56,72], so c_{11_2} [(56+60),(72+60)], that is c_{11_2}[116,132]. The state from M_{31} to M_{33}, $p_{11\_2}\{1 \to 0\}$, $p_{12}\{0 \to 1\}$, and the outcome is c_{12} [116,132]. From M_{33} to M_{35}, $p_{12}\{1 \to 0\}$, p_{13},$p_{22}\{0 \to 1\}$, c_{12}[116,132], so $c_{13}$and c_{22} both satisfy the time constraint [116,132]. From M_{35} to M_{37}, p_{18},p_{22} $\{1 \to 0\}$, $p_{19}\{0 \to 1\}$, we have known in the case l_{18} [0,120] and l_{22} [12, ∞], meanwhile c_{18} [0,0],

c_{22} [116,132]. p_{19} occurs after p_{18}, so c_{19} [(0+0),(0+120)] and p_{19} happens after p_{22}, get the result c_{19}[(12+116), ∞], that is c_{19}[128, ∞]. So c_{19} meets $\begin{cases} 0 \le c_{19} \le 120 \\ 128 \le c_{19} \le \infty \end{cases}$, and the two expressions conflict ($128 \le c_{19} \le 120$). The verification shows that the structure of the model is correct, but the time choreography has an unreasonable situation.

VI. Related Work

Remco M. Dijkman analyzes BPMN models using Petri nets, map the core elements of BPMN to Petri net, then call the tool, ProM, for static analysis of the model[1]. However, it only supports the simple test, can't verify the infinite loop and time properties. There are some tools based on Petri nets used for analyzing BPEL[3,4]. Reference [7] translate the BPMN to BPEL, then using Petri net to analyze the BPEL model. However, the types of verification problems for BPEL are different from BPMN in a large measure because of the block-structured nature of BPEL's control-flow constructs. And it increases a conversion step to BPEL, the efficiency must be decline.

Eder checks whether the process may complete before the deadline[9]. Son computes the time of tasks on the key path in order to deduce the deadline of the activity[10]. These tests are on the basis of the execution time of services has been known. This paper focuses on verifying BPMN model before the execution, and at this time we don't know the executed time of services. The time property in this paper refers to the expected time constraints, using it we test the compatibility of the model's time choreography. Nawal Guerrnouche uses the clock ordering process to discover time conflicts[12]. But the analysis of time constraints only caused by messages interaction, and the services in a process must be sequence structure. Obviously, it is not enough for the time choreography in terms of this paper.

VII. Conclusion

In this paper, we discuss the trustworthy composite service through business process model verification. We use time Petri nets as a basis for verifying BPMN models. Provide a mapping from BPMN models to time Petri nets, construct the reachability graph of the Petri net, and test the structure of the model on the basis. Furthermore, verify whether the model has time conflicts coming from incompatible time choreography. If the BPMN model becomes more complex, the corresponding time Petri net will be more complex. We need to consider simplifying the model before further verification, which is one of our next works.

ACKNOWLEDGMENT

This work was supported by the National High Technology Research and Development Program of China (863 program) under grant 2007AA010301, 2006AA01A106 and 2009AA01Z419.

References

[1] Remco M. Dijkman, Marlon Dumas, and Chun Quyang, "Semantics and analysis of business process models in BPMN," Information and Software Technology 50, 2008

[2] W.M.P. van der Aalst, and A.H.M. ter Hofstede, "Verification of workflow task structures: a Petri-net-based approach," Information Systems 25(1) ,2000, pp. 43−69.

[3] S. Hinz, K. Schmidt, and C. Stahl, "Transforming BPEL to Petri nets, in: Proceedings of the International Conference on Business Process Management," Lecture Notes in Computer Science, vol. 3649, Springer, 2005, pp. 220–235

[4] Chun Ouyang, Eric Verbeek, Wil M. P. van der Aalst, Stephen. Breutel, Marlon Dumas, and Arthur .H.M. ter Hofstede, "Formal semantics and analysis of control flow in WS-BPEL," Science of Computer Programming 67 (2–3) ,2007, pp. 162–198

[5] Xinming Ye, Jiantao Zhou, and Xiaoyu Song b, "On reachability graphs of Petri nets," Computers and Electrical Engineering 29, 2003, pp. 263−272

[6] Guermouche, N., and Godart, C, "Timed model checking based approach for compatibility analysis of synchronous web services," Research report, 2008

[7] Yida Lin, BPMN based Business Process Model Validity Analysis-A Petri Net Approach. Master's degree paper, National Taiwan University of Science and Technology. 2004

[8] Guermouche, N., Godart, C. Timed model checking based approach for compatibility analysis of synchronous web services. Research report, 2008

[9] Eder, J., and Tahamtan, A., "Temporal conformance of federated choreographies," Bhowmick, S.S., Kung, J., Wagner, R. (eds.) DEXA 2008. LNCS, vol. 5181, pp.668–675. Springer, Heidelberg, 2008

[10] SON J H, and KIM M H, "Improving the performance of time constrained workflow processing," Journal of Systems and Software, 58,2001, pp. 211-219.

[11] Furfaro A. and Nigro L., "Model Checking Time Petri Nets: A Translation Approach based on Uppaal and a Case Study," In Proceedings of IASTED International Conference on Software Engineering (SE 2005), Innsbruck, Austria, Acta Press,2005.

[12] Nawal Guermouche, and Claude Godart, "Asynchronous Timed Web Service-Aware Choreography Analysis," CAiSE 2009, LNCS 5565, pp. 364-378.

[13] Business Process Modeling Notation V1.1. http://www.omg.org/spec/BPMN/1.1/PDF OMG Document Number: formal, 2008

[14] Murata T, "Petri nets: properties, analysis and applications," Proc IEEE 77(4), pp. 541–80, April 1989