Substantiation of Web Service Composition Using Automata with Colored Petri Nets

Miss. Pratiksha S. Nandeshwar
M.E, 2nd Year: Information Technology Department
Siddhant College of Engineering, Pune, India
Email-id: nandeshwarpratiksha@gmail.com

Prof. Rashmi Deshpande
Assistant Professor, Information Technology Department
Siddhant College of Engineering, Pune, India
E-mail-id: rashmi2810@gmail.com

Abstract -- **At present, the web services are frequently used in web program development. Each service is independent in nature and these service agents may surround a nested call to other agents in order to absolute their tasks. Therefore, the meting out time of these agents may consume the meting out time according to the external process of others. Moreover, there might be some errors within the external web agents. Therefore, the web service flows must be determined before real implementation. As a consequence of such requirement, the Web Services Composition (WSC) must be clearly defined the flows and web descriptions to support the verification process of the web services agents. In this paper, we propose a new composition framework. We use automata to describe behaviors of web services. Each of underlying web services can interact with others through asynchronous messages passing according to its interaction role (client or server), with extending Web Services Definition Language (WSDL) of the nested schema. Furthermore, we provide the techniques to analyze and validate the CPNets and discover equivalent meanings of the properties in the context of Web services composition. Based on the model, the behaviors of a compound service can be simulated and validated to allow correcting the composition errors in advance, thus effectively averting it from runtime failure. Finally, we implement an example scenario to illustrate the effectiveness of our approach**.

Keywords- Web Service; Composition; WSDL; Automata; Verification; CPN

I. INTRODUCTION

Nowadays, distributed applications are increasingly being developed in the context of Service Oriented Architecture (SOA), where the basic unit of computation is called a service. According to W3C [2], a Web service is defined as a software system designed to support interoperable machine-to-machine interaction over a network. The technology basically articulates around the following three components: Simple Object Access Protocol (SOAP) provides the definition of an XML document, which can be used for exchanging structured and typed information between service peers in a decentralized distributed environment [3]. Web Services Description Language (WSDL) is an XML format for recounting network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to concrete network protocols and message formats to define an endpoint [4].Universal Description, Discovery and Integration (UDDI) focuses on the definition of a set of services supporting the description and discovery of the web services available for clients.

Model checking is one of the important methods to guarantee the system designing correctly. In Model checking based on automata i.e. MA is used for verification along with the CPN. It is mathematical model to represent the states behavior of the systems; set of all states visited infinitely often must be an element of the acceptance set. LTL is a model temporal logic with modalities referring to type and it is used to specify concurrent prosperity. Based on this, many researchers propose the model checking algorithm based on LTL.

Although there are some researchers using Petri nets to model services composition, they just analyzed the models not suitable for verification. Colored Petri Nets (CPNs) extend the classical Petri Nets with colors (to model data), time (to model durations), and hierarchy (to structure large models). Like in classical Petri Nets, CPNs use three basic concepts: transition, place, and token.

There are several reasons for modeling and verifying composite services using colored Petri nets. First of all, CPNs have formal semantics and allow for different types of analysis, e.g., state-space analysis and invariants. Second, CPNs are executable and allow for rapid prototyping and simulation. Third, CPNs are graphical. Finally, the CPNs language is supported by CPN Tools a graphical environment to model and analyze CPNs. This article proposes the method combing Model Checking with Colored Petri Net (CPN) to verify the services composition correctly.

In this paper, Initially Web Services are defined using WSDL and further composed using WS-CDL. Composed web services are mechanically compiled into the Finite State Process notation (FSP) to succinctly describe and reason about the concurrent programs. Implementations are mechanically translated to FSP to allow a trace equivalence verification process to be performed. By providing early design verification, the implementation, testing and deployment of web service compositions can be eased through the understanding of the differences, limitations and undesirable traces allowed by the composition. Finally, it modeled using CPN for automatic analysis and verification of the behavior of systems. We provide the techniques to analyze and validate the CPNets and discover equivalent meanings of the properties in the context of Web services composition. Based on the model, the behaviors of a compound service can be simulated and validated to allow correcting the

composition errors in advance, thus effectively averting it from runtime failure.

II. RELATED WORK

Currently, some service composition languages have existed, such as WSFL, WSC, WSCI, WS-BPEL, Petri net and so on. WS-BPEL has been proposed by OASIS as an industry standard and is supported by major software companies such as IBM, Oracle, and SAP, so it is becoming dominant. Some references [1, 2, 6] were suggested Colored Petri Net (CPN) [5] is an extension of ordinary Petri Net; it combines Petri Net with the programming language, and describes concurrent system briefly.

References [3,4], used Petri Net, a formal model based on Petri Net is one of commonly used service composition methods, proposes to use Petri Net to describe service, and gives modeling, analysis and verification of service composition. Petri Net is limited to describe control flow of service composition because of its structure, in order to be more specific description of service composition, we need to combine data flow and control flow.

Reference [7] suggesting that a mechanism called eFlow is a service composition platform based on workflow; it provides definition, creation and control of composition. Reference [8], proposed SWORD is the service composition system that uses ER (entity-relationship) model to create service. The composition protects have the common disadvantages that can't make analysis and verification of service composition process and guarantee the correctness of service composition.

References [9, 11, 12], suggested that timed automata are a mathematical model to represent the behavior of the system and mainly used to describe states and behavior of single activity. It is a Finite state machine extended with clocks. Uses a dense time model where a clock variable evaluates a real number, all the clocks progress synchronously. The timed automata have used the tool for verification of the service composition. This can be done by WST (Web Service Translator Tool). WSCDL converted into Timed Automata (TA) and finally verified using Uppaal Tool. Here the Reachability problem is verified. It does not cover the emptiness, deadlock, state explosion etc.

Reference [10], suggested Interface automata, it is a type of I/O Automata. Interact through synchronization of checks the compatibility. For composition BPEL4WS was used and mapped into Interface automata and transformed into Promela. Finally, it is verified using SPIN Tool [11], here correctness property is verified. It does not cover the emptiness, deadlock, dead transition, state explosion etc.

EMTI method [14], has been proposed using the extending framework, EEMTI, with extending Web Services Definition Language (WSDL) of the nested schema. It was limited for composition only, does not suggest the problems like reachability and state space.

Furthermore, although many tools can help to find the syntax errors of WS-BPEL, it is impossible to completely eliminate the logical errors in process. Therefore, it is necessary to create formal model for service composition process and realize the reliability of service composition that is one of the key issues to be solved.

III. PROPOSED WORK

This paper proposes a new method for composing web services by defining a "WSComposition profile" which is an extension of the WS-CDL metamodel. The proposed method transformed composition into automata diagram for early verification of invoked services. Therefore, this profile allows us to represent the behavioral characteristics of web services and provides an easy way to design and compose web services based on their behavioral aspect.

Basically, our approach consists of three main steps. . In the first step, existing simple web services are discovered and situated in the web service registry using WSDL. In the second step, the independent web services are composed using WS-CDL according to user requirement. Implementations are mechanically translated to FSP to allow a trace equivalence verification process to be performed. By providing early design verification, the implementation, testing and deployment of web service compositions can be eased through the understanding of the differences, limitations and undesirable traces allowed by the composition. Finally, it modeled using CPN for automatic analysis and verification of the behavior of systems. We provide the techniques to analyze and validate the CPNets and discover equivalent meanings of the properties in the context of Web services composition. Based on the model, the behaviors of a compound service can be simulated and validated to allow correcting the composition errors in advance, thus effectively averting it from runtime failure.

A. System Architecture

Proposed architecture consists of several steps: Initially, web services are developed independently using WSDL and composed using WS-CDL. We are using two different and sovereign approaches for verification.

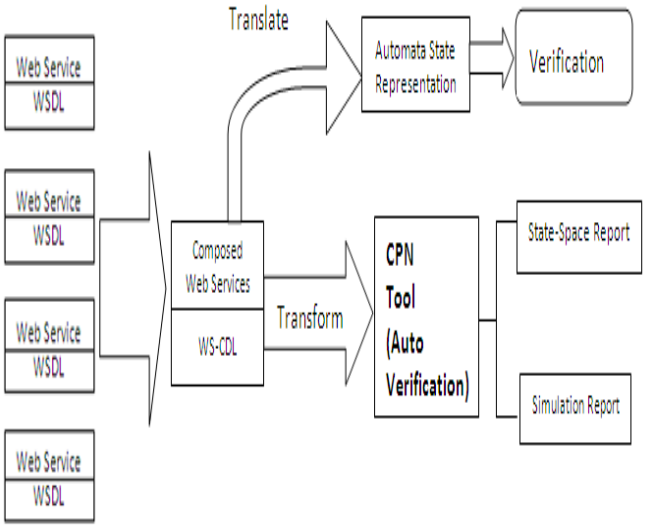

Figure 1: Proposed System Architecture

In one approach, the composed services are further transform into automata state diagram i.e. as the numbers of autonomous services are invoked, it will form state diagram and through state diagram we can easily verify the composition.

Another approach is based on an open source tool called as Colored Petri Nets (CPNs), in which all the web services are modeled and designed using CPN. CPN provides an automated mechanism for verification at design time, It also provides various analysis such as State-space and Simulation Report.

B. Sequence Diagram

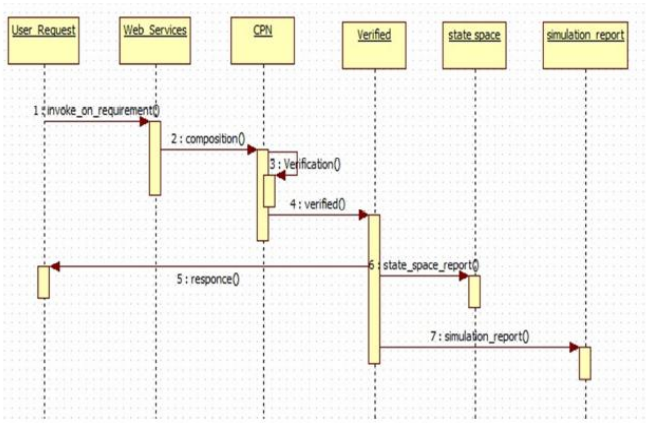

Figure 2: Sequence Diagram for CPN

Overall Sequence diagram shown in fig. 2, Web services are invoked according to user request and further composed and designed using CPN. Figure shows that CPN provides an automated verification to composed services, if it is correct then it would be sending to user as response. State –space and Simulation result is also generated through CPN.

C. Verification Algorithm

For verification of composed services Rechability and Emptyness two parameters are considered. Reachability is checking that our resultant node is reachable from initial node or not, Emptiness is help to check reachability condition. For example- suppose, the very next service is not reachable because of unavailability then through emptiness property we can move for further service.

In other word we can say that if reachability condition is satisfied than automatically emptiness condition will be satisfied.

Algorithm: Rechability & Emptiness

Initialize path->ø, F= *F* (F1, F2, F3… Fn)
Webverification ((Q, ∑, Is,q0, F), path)
begin
path. Put(q0)

Reachabilitysearch (Finalstate f)
begin
while (path is not empty) do begin
q0->q1
path. Put(q1)
for all(*F*) begin
if(f in *F*) begin
bool reach=”true”
exit
end
end
end
if(reach is true) begin
f is reachable
end
else begin
f is not reachable
end

emptinesssearch() begin
while (path is not empty) do begin
q0->q1
path. Put(q1)
if(q1 is not empty & q1 is in *F*) then
begin
no emptiness problem for reaching *F*
end
else if (q1 is empty) then begin
emptiness problem
end
end
end

D. Case Study

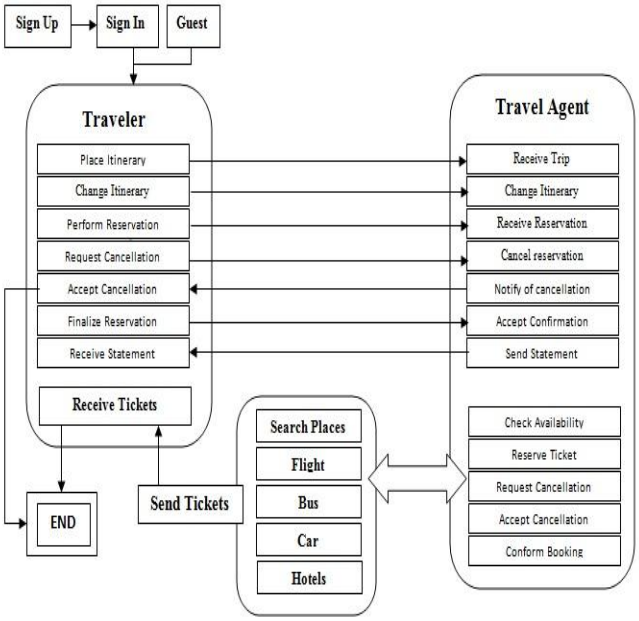

Figure 3: Plan and Book Trip

The fig. 3, "Plan and Book Trip" process consists of three participants: a Traveler, a Travel Agent and search & Reservation System. The Traveler planning on taking a trip decides the destination and calendars, and submits its choice to a Travel Agent. The Agent then finds the best travel plan and asks the Airline and (or) Bus and (or) Car and (or) Hotels to verify the availability of seats. And the Reservation system provides information about the

availability of seats. Thereafter, the traveler provides the Agent with her Credit Card information to properly book the seats based on their request. The Traveler also provides his/her contact information to receive an e-Ticket. Then the Reservation system books the seats and issues e-tickets to the Traveler. Finally, the Agent charges the Traveler's Credit Card and sends the notification of the charge.

The following figure 4, demonstrates an example of how the web services composition specification is transformed into a CPNs model on the basis of the transformation rules. The individual Interface Net (I-Net) is generated firstly, and then different I-Nets are integrated into a composite net. The overall CPNs model of the use case "*Plan and Book Trip*" is constructed with CPN Tools after a number of simplified assumptions and abstractions.

E. CPN Model for "Plan and Book Trip"

The type of tokens that may reside in a place is determined by the color set of the place. A color set in a CPNs model is similar to a data type in a programming language, and the values in a color set are referred to as colors. The color set of a place is typically written below the place and is declared using the Standard ML [9] programming languages.

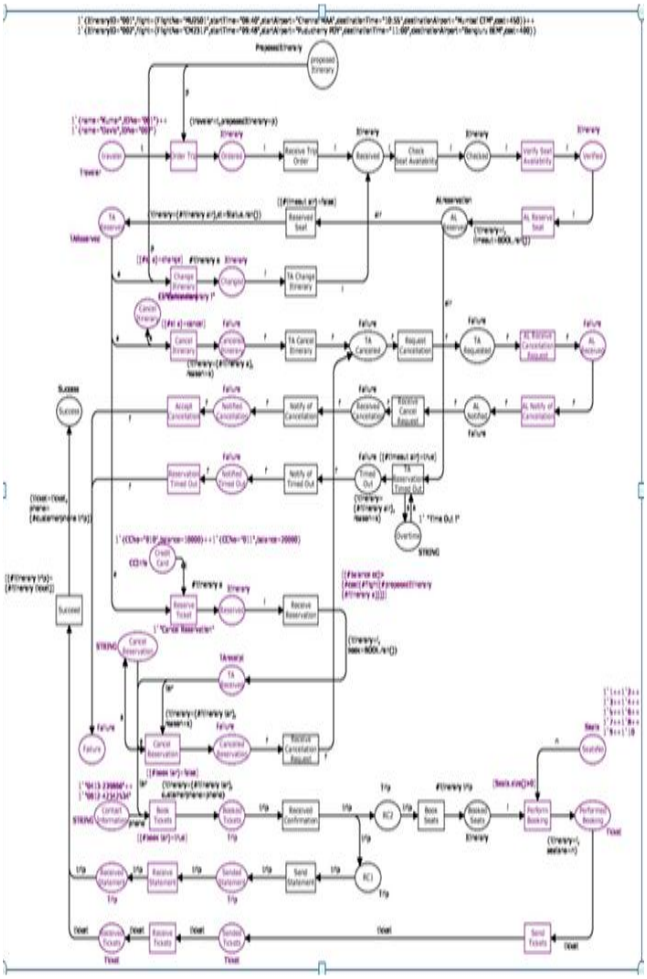

Figure 4: CPN Model for "Plan and Book Trip"

```
(* Standard priorities *)
        val P_HIGH = 100;
       val P_NORMAL = 1000;
        val P_LOW = 10000;
(* Standard declarations *)
       colset UNIT = unit;
       colset INT = int;
       colset BOOL = bool;
       colset STRING = string;
        fun positive(i)=(i>0);
       colset PositiveINT=subset INT by positive;
                colset Traveler=record name:
       STRING*IDNo:STRING;
        colset Flight=record
       FlightNo:STRING*startTime:STRING*
       startAirport:STRING*destinationAirport:
       STRING*
        destinationTime:STRING*cost:PositiveINT;
                 colset ProposedItinerary=record
       itineraryID:STRING*flight:Flight;
       colset Itinerary=record traveler:Traveler*
        proposedItinerary:ProposedItinerary;
                 colset ALreservation=record
       itinerary:Itinerary*timeout:BOOL;
       colset Status=with change|cancel|reserve;
                colset TAReserved=record
       itinerary:Itinerary*st:Status;
        var t:Traveler;
       var p:ProposedItinerary;
       var i:Itinerary;
       var phone:STRING;
       var alr:ALreservation;
                colset Trip=record
       itinerary:Itinerary*customerphone:STRING;
       var trip:Trip;
                colset Failure=record
       itinerary:Itinerary*reason:STRING;
       var f:Failure;
                 colset CCInfo=record
       CCNo:STRING*balance:PositiveINT;
        var cc:CCInfo;
                colset TAreceipt=record
       itinerary:Itinerary*book:BOOL;
        colset Seats=int with 1..10;
       var n:Seats;
colset Ticket=record itinerary:Itinerary*seatsno:Seats;
var ticket:Ticket;
colset Success=record ticket:Ticket*phone:STRING;
colset Timeout=record itinerary:Itinerary;
var tar:TAreceipt;
var a:TAReserved;
var sc:Success;
var to:Timeout;
var s:STRING;
```

Figure 5: Color Sets, Functions and Variables Declarations in CPN

Fig. 6(a) and 6(b), describes two different automata state diagram that are generated through the user requests. In First 6(a) shows only two services are invoked "search places" and "flight". In 6(b), more than two numbers of services are invoked by user. In both figure after invoking the services, payment service is invoked. That is calculated sum of request(s) sent by customer or user.

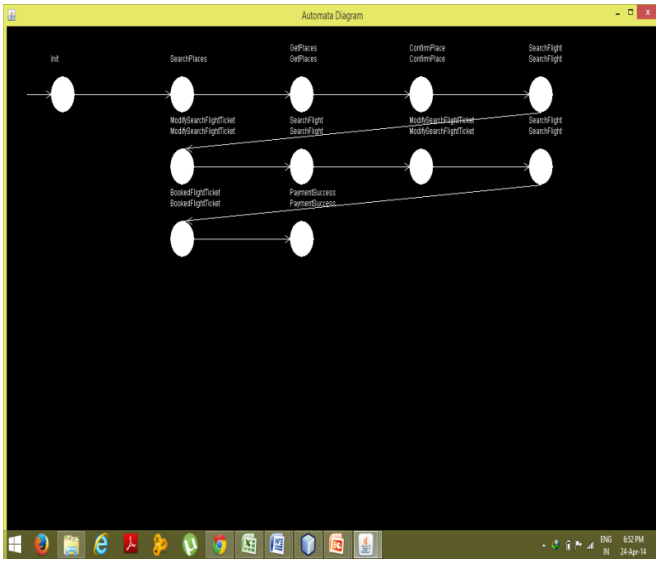

Figure 6(a): State Diagram for "Plan and Book Trip"

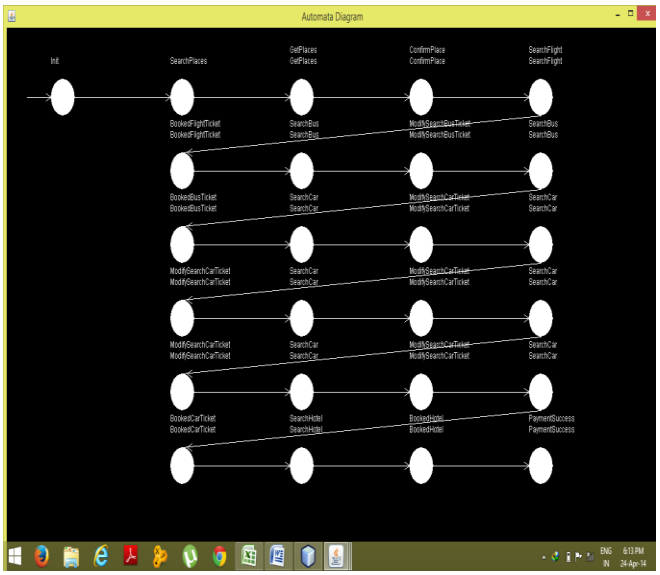

Figure 6(b): State Diagram for "Plan and Book Trip"

F. Implementation Results and Performance Metrics

a) State-Space and Simulation Report

1. *State-Space Report*

Statistics
State Space
Nodes: 75196
Arcs: 211908
Secs: 3409
Status: Full

2. *Simulation Report*

Succeed @ (1:Flight_Reservation_ System) - trip = {itinerary={traveler= {name= "David", IDNo="002"} ,proposedItinerary={ItineraryID="001", flight={Flight No="MU2501", Start Time= "08:40", startAirport="Chennai MAA",destination Time="10:55", destinationAirport = "Mumbai CTM" , cost=450}}} ,customer phone= "0413-239866"} - ticket= {itinerary={traveller ={name ="David", IDNo="002"} ,proposedItinerary= {ItineraryID= "001", flight={FlightNo=" MU2501", startTime ="08:40",startAirport= "Chennai MAA",destinationTime ="10:55",destinationAirport ="Mumbai CTM",cost=450 }}} ,seatsno=10}

b) Performance Analysis

$$RF_{TA} = \frac{\sum (FS\text{-}TOS)}{Q} \times 100$$

$$RF_{MA} = \frac{\sum (FS_{ta}\text{-}Esm\text{-}ToS)}{Q} \times 100$$

Where, RF_{TA} = Reachability factor of Timed Automata
RF_{MA} = Reachability factor of Muller Automata
FS = Final State
TOS = Timed Out States
Esm = Empty state moved
Q = Total number of states

For analysis of reachability factor, approx 50 services are implemented and compared proposed approach with the existing approach. Using above mentioned formula calculated data are shown in Table 1.

TABLE I. RESPONSE TIME AND REACHABILITY FACTOR FOR TIMED AUTOMATA AND MODIFIED AUTOMATA

S. No.	Number of Web Services	TA		MA	
		RF_{TA}	RT_{TA}	RF_{MA}	RT_{MA}
1	8	88	10	63	6
2	12	92	25	50	10
3	16	94	36	50	18
4	20	90	42	55	22
5	30	93	56	60	27
6	38	95	65	63	32
7	42	93	73	60	38
8	48	94	82	61	42

These above data are shown in graphical form in fig. 7 and 8. From comparison graph we clearly conclude that, our proposed work is providing better result as compare to existing one.

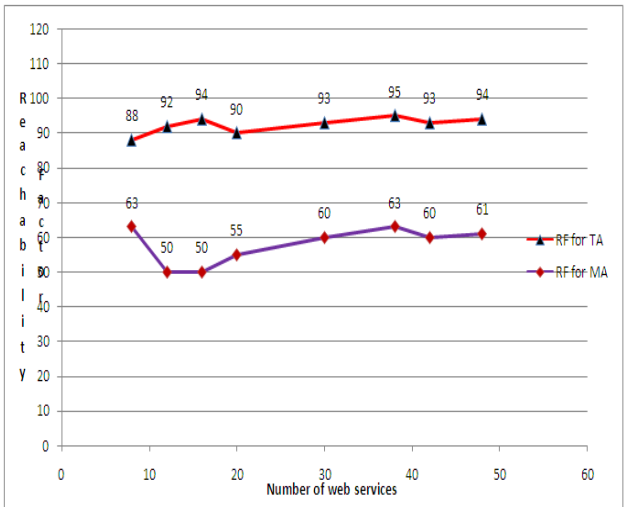

Figure 7: Rechability Factor Comparison

Response time is directly proportional to reachability factor, that means if reachability factor is high than it will take more time for completion of work or if it is less than it will take more time for completion of work.

$$RT \quad \alpha \quad RF$$

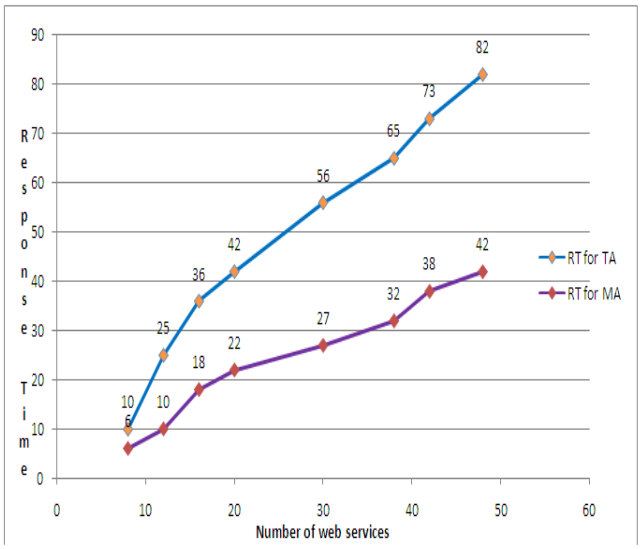

Figure 8: Response Time Comparison

IV. CONCLUSION

In this paper, we have focused on two different aspects of web services, composition and verification. A new algorithm is proposed for better verification of web services. CPN tool is used for design and analysis of system, through analysis various performance metrics are discussed and some problems has been identified ex. state-space. Mainly, two performance metrics are taken into account as reachability factor and response time. Finally, we compared experimental results of our work with existing systems. and we conclude that our work giving better performance.

CPN tool works on standard Modeling Language (ML) only. without the knowledge of ML, CPN cannot be implemented for any kind of application. In future our plan to design a user friendly new tool for design, analyze and verify of system, also our plan to implement our work into tool for better performance of system.

REFERENCES

[1] J. Zhu et al. "Verifying Web Services Composition based on LTL and Colored Petri Net", IEEE, The 6th International Conference on Computer Science & Education (ICCSE 2011), SuperStar Virgo, Singapore, pp. 1127-1130, August 3-5, 2011.

[2] X. Deng et al. "Modeling and Verifying Web Service Composition Using Colored Petri Nets Based On WSCI", Proceedings of the IEEM, IEEE, pp. 1863-1867, 2007.

[3] P. Xiong, et al. "A Petri Net Approach to Analysis and Composition of Web Services", IEEE Transactions on systems, man, and cybernetics—part a: Systems and Humans, vol. 40, no. 2, pp.376-387, march 2010.

[4] R. Hamadi and B. Benatallah, "A Petri Net-based Model for Web Service Composition", Fourteenth Australasian Database Conference (ADC), Adelaide, Australia. Conferences in Research and Practice in Information Technology , Vol. 17, , pp. 191-200, 2003.

[5] K. Jensen et al. "Coloured Petri Nets and CPN Tools for modeling and validation of concurrent systems", Springer-Verlag 13, pp. 213-254, March 2007.

[6] X. Yi and K. J. Kochut, "A CP-nets-based Design and Verification Framework for Web Services Composition", Proceedings of the IEEE International Conference on Web Services (ICWS'04), pp. 756-760, 2004.

[7] F. Casat, S. and I. Krishnamoorthy, "Adaptive and Dynamic Service Composition in eFlow[EB/OL]", Software Technology Laboratory HP Laboratories, 2000.

[8] S. R. Ponnekanti and A. Fox, "Sword: A Developer Toolkit for Web Service Composition[EB/OL]", In Proceedings of the Eleventh World Wide Web Conference, pp.83-107, 2002.

[9] M. E. Cambronero et al., " Validation and verification of web services choreographies by using timed automata", The Journal of Logic and Algebraic Programming, 80 (2011) , Elsevier, pp.25-49, 2011.

[10] H. Su. et al., " Interface automata based formal model for BPEL4WS Web services composition", Application Research of Compliers (in Chinese). vol. 26, 2009, pp. 1774-77.

[11] G. Holzmarm, "The model checker SPIN", IEEE Transactions on Software Engineering. vol. 1, pp.279-95, 1997.

[12] S. Dong et al., "Verification of compilation orchestration via timed automata", International Conference on Formal Engineering Methods (ICFEM), pp. 226-45, 2006.

[13] X. Wang et al, "An Interface Automata based Model for Web Service Composition", pp.945-950, IEEE 2008.

[14] Nalinrat Srirajun et al. "EEMTI: an Extending Framework for Nested Web Service Verification", pp. 128-133, IEEE, 2012.