ISSN No. 0976-5697

Volume 2, No. 1, Jan-Feb 2011

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

Simulation Based QoS Analysis of Service Composition

Mojdeh Rahmanian
Young Researchers Club, Department of Computer
Islamic Azad University, Jahrom Branch
Jahrom, Iran
m.rahmanian@jia.ac.ir

Abstract: In recent years, various organizations offer their services with independent and reusable programs on internet. An important challenge is to integrate them to create new value-added Web services. The selection of the most appropriate candidate for each service for complex composite Web services becomes a difficult task. One possible solution is to use quality of service (QoS) to evaluate, compare and select the most appropriate composition. Typical QoS properties associated with a web service are the execution cost and time, availability, Throughput. We use Coloured Petri Nets (CPNs) for Web services composition. CPNs combine the strengths of Petri nets with the functional programming language Standard ML. We use the associated software tool called CPN Tools to perform the analysis. We use the associated facilities of CPN Tools to perform evaluate the QoS metrics.

Keywords: web service; service composition; Coloured Petri Net; Quality of Service

I. INTRODUCTION

Service oriented architecture (SOA) [1,2,3], are software applications that allow the exchange of information over a network between different software entities [4]. Services are reusable, autonomous and self descriptive application programs that can be accessible via standard network protocols such as but not limited to SOAP[5] over HTTP. Services can be presented on a network of computers with different operating systems platforms as well as various programming languages.

In SOA, every person or organization can be a service provider or a service requester and the communication between them is achieved with XML messages and SOAP protocols. Services specifications are stored in a repository called UDDI[1][6]. The language that is used for this specification is WSDL[2] [7].

Web service composition is a task of combining and linking existing web services to create new web processes in order to add value to the collection of services. Using of visual and descriptive tools that can quickly and easily model services and their compositions as well as analyzing them before implementing, is necessary. For the sake of fast computation, many researchers prefer Petri nets [8], since they are well suited for capturing flows in web services, modeling the distributed nature of web services, representing methods in a web service and reasoning about the correctness of the flows. A web service behavior is basically a partially ordered set of operations. Therefore, it is straightforward to map it into a Petri net.

The system should be able to detect quality patterns and variations and adjust its selection accordingly. Quality of Service (QoS) refers to the non functional aspects of Web services. Several attributes can be taken into consideration at once. In the case where several services are supposed to execute in parallel, finding the best solution is not as easy as choosing the best individual service but rather choosing the service that would perform best as part of the complete composition. Depending on the user's need, several quality attributes might be of importance, such as cost, response time, availability, reliability, security, throughput, reputation and so forth.[9]

In this paper we introduce a Colored Petri Nets (CPNs) based approach to modeling and analyzing web service compositions.

This work is organized as follows. Section II presents related work. Section III gives a brief overview of CPNs. The base services are introduced in Section IV. In section V presents the QoS attributes for web service composition. Our web service composition examples are explained in Section VI. The CPN model is built in Section VII and constructed with CPN Tools in Section VIII. Numerical results are given in Section IX. Finally, Section X concludes and gives some perspectives to this work.

II. RELATED WORK

Over the years there have been several research efforts to model and analyze Web services. In [10], a number of operators for proposing different compositions of web services were given and formal meaning of these operators is shown using Petri-Nets. Any relation among services shown as an expression of these operators can be converted to a model in Petri-Nets. Also, by using several features for these operators, it is possible to transform and improve relationships between them in such a way that their initial properties be unchanged. In [11] describes new facilities that are fully integrated with CPN Tools and supports simulation-based performance analysis using CP-nets. Also, by using a simple example of a network protocol, it illustrates how one can use these facilities to collect data during simulations, for generating different kinds of performance-related output, and for running multiple simulation replications. In [12], DAML-S service descriptions of composite services are represented as Petri Nets (Petri 1962), providing decision procedures for Web services simulation, verification and composition.

[1] Universal Description, Discovery and Integration
[2] Web Service Description Language

III. COLORED PETRI NETS

Petri-Net [13, 14, 15] is a graphical and mathematical tool, used for specification and study of concurrent, asynchronous and/or distributed information processing systems. Since service oriented systems can have these features, using Petri-Nets for specification of these systems is appropriate.

A Petri-Net is a directed and connected graph which has three components :(1) Nodes which can be either a place or a transition. (2) Arcs always connect a place to a transition or a transition to a place. It is illegal to have an arc between two nodes of the same kind, i.e., between two transitions or two places (3) Tokens that are placed in places. If at least one token exists in every input place which connects to a transition, that transition is called "enabled". Enabled transition can "fire", so that a token would be omitted from each input place of it, and a token would be placed in each output place of it. The use of visual modeling techniques such as Petri-Nets in the design of complex Web services is justified by many reasons. For example, visual representations providing a high-level yet precise language allows to express and reason about concepts at their natural level of abstraction. A Web service behavior is basically a partially ordered set of operations. Therefore, it is straight-forward to map it into a Petri-Net. Operations are modeled by transitions and the state of the service is modeled by places. The arrows between places and transitions are used to specify causal relations. [10] Thus, firing of a transition that causes moving tokens from some places to others, models an operation that changes the state of a system.

A. *Formal definitions*

Definition 1. "A Petri-Net is a 5-tuple,
PN = (P, T, F, W, M0) where:

- P = {p1,p2, ….,pm} is a finite set of places,
- T = {t1,t2,….,tm} is a finite set of transitions,
- $F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs (flow relation),
- W: $F \rightarrow \{1,2,3,\ldots\}$ is a weight function,
- M0: $P \rightarrow \{0,1,2,3,\ldots\}$ is the initial marking,
- $P \cap T = \varnothing$ and $P \cup T \neq \varnothing$". [15]

Timed Petri-Net (TPN) is a Petri-Net in which timed places or timed transitions exist. According to definition 1, a formal definition for a TTPN can be stated as:

Definition 2. A Petri-Net is a 6-tuple, TTPN = PN ∪ LT where:

- PN is a Petri-Net, and
- LT: $T \rightarrow D$ is latency time function of transitions. D, set of transitions latency numbers, is a set of numbers that each of them is the latency time number of a transition. This number shows that how many units of times after enabling, the transition will fire.

Input place is a place that does not have input arc and when a token is placed on that place, it will mean the service has received the necessary information from environment for its operation and is in the ready state. Output place is a place that does not have output arc and when a token is placed on that place, it will mean that the service has returned the results of its operation to environment and is in the complete state.

Colored Petri Nets (CPNs) are extensions of Petri Nets that allow modeling of both timed and untimed and support a powerful module mechanism that allows building of models in hierarchical manner.

CPNs is a discrete-event modelling language combining Petri nets [8] and the functional programming language CPN ML which is based on Standard ML [16,17]. The inscriptions are written in the CPN ML programming language which is an extension of the Standard ML language.

IV. BASE SERVICES

Each service can implement a specific operation that it is developed for it. However, to reply to a majority of requirements, a process must be done that for implementing it, several base services must be composed. For implementing different processes, base services communicate and coordinate with each other in different shapes, each shape appropriate for the process being implemented. Hence for implementing a process, a composite service will be developed by composing base services.

We assume that necessary base services for the compositions are ready; so, we should only concentrate on how to compose them. The base services, S1,…, S5 are shown in Fig. 1. As shown in the Fig. 1, a base service is represented by a rectangle and only those nodes that play a role in communication between services is shown in its figure. Different shapes considered for these services are to distinguish between them in a composition and is in some cases because of their different roles which must be played in compositions.

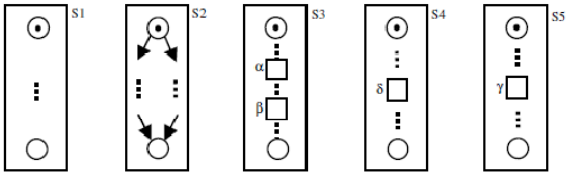

Figure 1: Base Services

V. QOS ATTRIBUTES

Individual Web services are conceptually limited to relatively simple functionalities modeled through a collection of simple operations. However, for certain types of applications, it is necessary to combine a set of individual web services to obtain more complex ones, called composite or aggregated web services as mentioned in previous section. One important issue within Web service composition is related to the selection of the most appropriate one among the different possible compositions. One possible solution is to use quality of service (QoS) to evaluate, compare and select the most appropriate composition(s). [18] Quality of Service (QoS) refers to the non functional aspects of Web services.

The area of QoS management covers a wide range of techniques that match the needs of service requesters with service providers. service provider provides differentiated Web services to different customers. In this research, the following four quality metrics of a Web service are considered

- **Response Time**: the time interval between when a service is invoked and when the service is finished.
- **Execution Cost**: the price that a service requester has to pay for invoking the service;
- **Execution Time**: the average time expected for executing a web service.
- **Availability**: the probability that the service is available at some period of time;
- **Throughput**: the number of completions of web service requests during an observation time interval.

VI. DYNAMIC SERVICE COMPOSITION

To illustrate the use of CPN and CPN Tools in modeling and analyzing web services we consider a simple case as shown in Fig. 2 which has consumers and one provider, where the objective is to do a QoS analysis of various options available for service composition and orchestration. In the first composition, the path would have to execute *Provider1*, taking only 50 milliseconds (ms) to complete and costing $10 with limited service availability.

In the second one, the first path would have to execute 2 services, (*Provider1* and *Provider2*), which *Provider2* taking only 250 milliseconds (ms) to complete and costing $30 and unlimited service availability. In case of no enough resources available in *Provider1*, we can use of a *Provider2*.

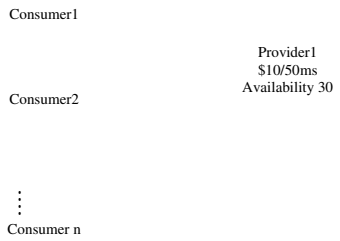

Figure 2. Simple Service Composition 1

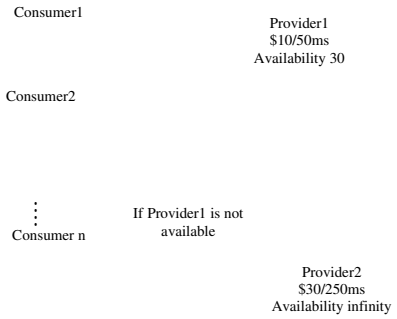

Figure 3. Simple Service Composition 2

We can consider it as a online library system where we can view customers as consumers. If the books are not available in *Provider1* then the request can be placed to another library which is the *provider2*, which can either accept the request or decline it based on the availability. In fact, We have a choice of making use of a third-party service provider for some of the functionality we want to create. But this option has an additional cost per use. Before coding and deploying our services, we would like to evaluate each of these options in terms of QoS attribiute. It is assumed that for our customer base, we have a lower Execution Cost whereas making use of third-party contracted services definitely has a higher delay factor and higher Execution Cost. It is also assumed that the thirdparty provider has better and more computational resources.

VII. THE CPN MODEL

Petri Nets models have been a useful formalism in the information technology industry, which are capable of presenting complex system processes. This study develops a Petri Nets-based Composing platform to address the issues of composing Web services.

CPNs support a powerful module mechanism to support hierarchical construction of a model. The module concept of CP-nets is based on a hierarchical structuring mechanism, allowing a module to have submodules and allowing a set of modules to be composed to form a new module. [19] The detailed activity associated with each of these entities can then be represented on a subpage associated with it. Such transitions are called substitution transitions. In CPN Tools, a substitution transition is shown with a double border. The state of the modelled system is represented by the places. Each place can be marked with one or more *tokens*, and each token has a data value attached to it. This data value is called the *token colour*. It is the number of tokens and the token colours on the individual places which together represent the state of the system. This is called a *marking* of the CPN model, while the tokens on a specific place constitute the marking of that place.

Next to each place, there is an inscription which determines the set of token colours (data values) that the tokens on the place are allowed to have. The set of possible token colours is specified by means of a type (as known from programming languages), and it is called the *colour set* of the place. By convention the colour set is written below the place.

Based on our first model in Fig. 2, we have Consumer, one Provider and the service request and response messages flow from the provider to the consumer and provider via the network as shown in Fig 2.

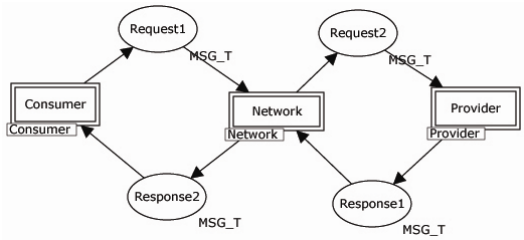

Figure 4: top-level module of the hierarchical composition model.

In this paper, each service requests and responses carry a number which denote the service request and the response is the same as the service number. In addition to service number, the meesages carry *Execution Time* related to each service providers. In CPN, the (type) declaration for tokens is given using a color set. The relevant color set declarations for messages are given in Fig. 5.

```
colset SID = product ID * COST;
colset MSG = union R: SID;
colset MSG_T = MSG timed;
colset MSGxTime = product MSG * TOR;
```

Figure 5: Colour sets for the timed CPN model

For simulation-based performance analysis and exploring performance measures such as response time and throughput,

we need the time concept. In CP-Nets the passage of time is modeled by *global clock*. In the timed CPN models, tokens carry a *timestamp* denoting the earliest (model) time at which the token can be used, i.e., removed by the occurrence of binding element.

The activities associated with a consumer are modeled in the subpage Consumer shown in Fig. 6. In this net, the place *Ready* contains a timed token. The time stamp of this token denotes the earliest time the next request will be sent. A service request is sent by firing the transition *SendRequest*. The generated request (token) consists of a request identifier and a service and added to place *Request1* denoting sending of the request into the network. The place *NextID* keeps track of the next request number. Each time a request is sent, this number gets incremented by one. This is denoted by the increment expression on the associated arc. Once a request is sent, a token consisting of the time of the request and the request itself is added to place *Wait* denoting the activity of waiting for rely. In this simplified system, we do not model any timeouts. Thus, a request may wait for arbitrary long time for a response. When a response is received from the network, that is, a token appears in the place *Response2*, it is matched for the id numbers before it is accepted by the firing of the transition *ReceiveResponse*. This is controlled by the guard [i1=i2] associated with this transition. In general, a guard may be any arbitrary sequence of boolean expressions belonging to the underlying programming language CPN ML which itself is based on the popular functional language SML [17].

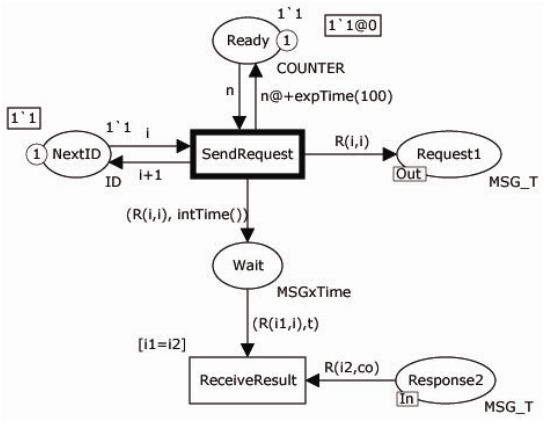

Figure 6: Consumer Module

We have assumed the network to operate under ideal conditions. That is, we assume Packet loss and retransmission issues are also not accounted, because the model becomes too specific to the protocol implementation. The server is only concerned with servicing a number of HTTP requests generated by the consumers. If we want to change these assumptions, we could take advantage of the modular approach supported by CPN and enhance the network behavior without affecting the other components of the system. The ideal network under the present assumptions is depicted in Fig. 7.

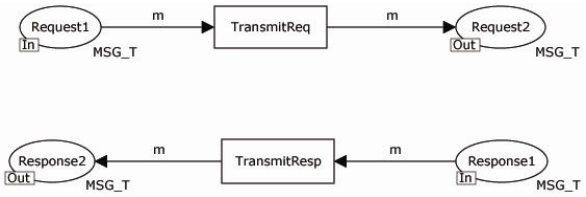

Figure 7: Network Module

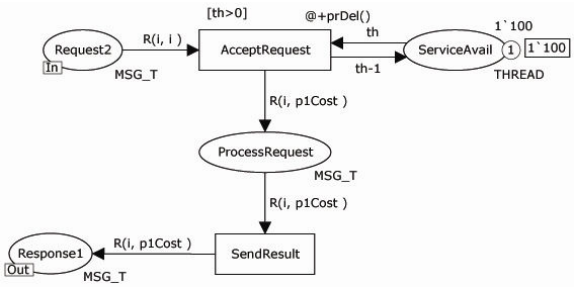

Figure 8: Provider Module

The final component of our model is the *provider1*. Quality of service annotations are encoded as color set. The Availability of services represented by the place *ServiceAvail*. In the net depicted in Fig. 8, a request received from the network (i.e., a token in place Request2) is accepted and serviced only if there are enough resources available.

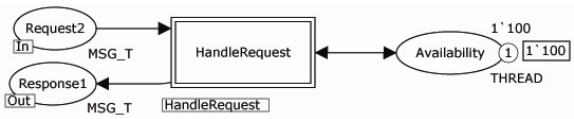

Figure 9: Provider Module in Second Model

In the second model as shown in Fig. 9, the consumer part is similar to the Fig. 6. The provider side in this one, if there is no resource available, the request is sent to *provider2*. In Fig. 9, the substitution transition *HandleRequest* choose if the availability is zero in *provider1* send the request to *provider2*.

Before implementing and deploying our services, we would like to evaluate each of these options in terms of Response Time and throughput.In the next step we develop our CPN model to evaluate these scenarios.

VIII. CONSTRUCTION OF CPN MODELS

CPN Tools [20] is a graphical software tool that supports modification, simulation, state space analysis, and performance analysis of CPN models. While constructing a model, syntax checking is incrementally performed by the CPN tool. CPN Tools allows one to create a visual representation of a CPN model which is based on interaction techniques, such as tool palettes and marking menus. It is possible to explore the behavior of the modeled system using simulation. A license for CPN Tools can be obtained via the CPN Tools web site [20]. Readers are recommended to refer to [21] for further information and guidelines for CPN tools.

A simulation of the composition models shown in Fig. 10. During simulating of a model we can examine and extract

information from the states and events without having to modify the structure of it. In CPN Tools it is possible with monitors. A monitor is a mechanism that is used to observe, inspect, control or modify a simulation of a CP-net. [22] CPN Tools support different kinds of monitors. Depends on our purpose we can use one kind of them. In our work, we use *monitors* for calculating QoS metrics of web services; Response Time and Throughput. In CPN Tools, monitors can be used to examine the binding elements that occur and the markings that are reached during a simulation. Different kinds of monitors can be used for different purposes [19].

The Response Time represents the time to send a request from the consumer to the provider i.e. when SendRequest transition occurs and back to the consumer i.e. when the ReceiveResult fires.

We apply a data collector monitor on *ReceiveResult* transition for collecting specific data during steps of simulation. The Type of the monitor indicates that it is a data collector monitor, i.e. it collects the numerical data on our transition. The transition *ReceiveResult* that is associated with the monitor can be found

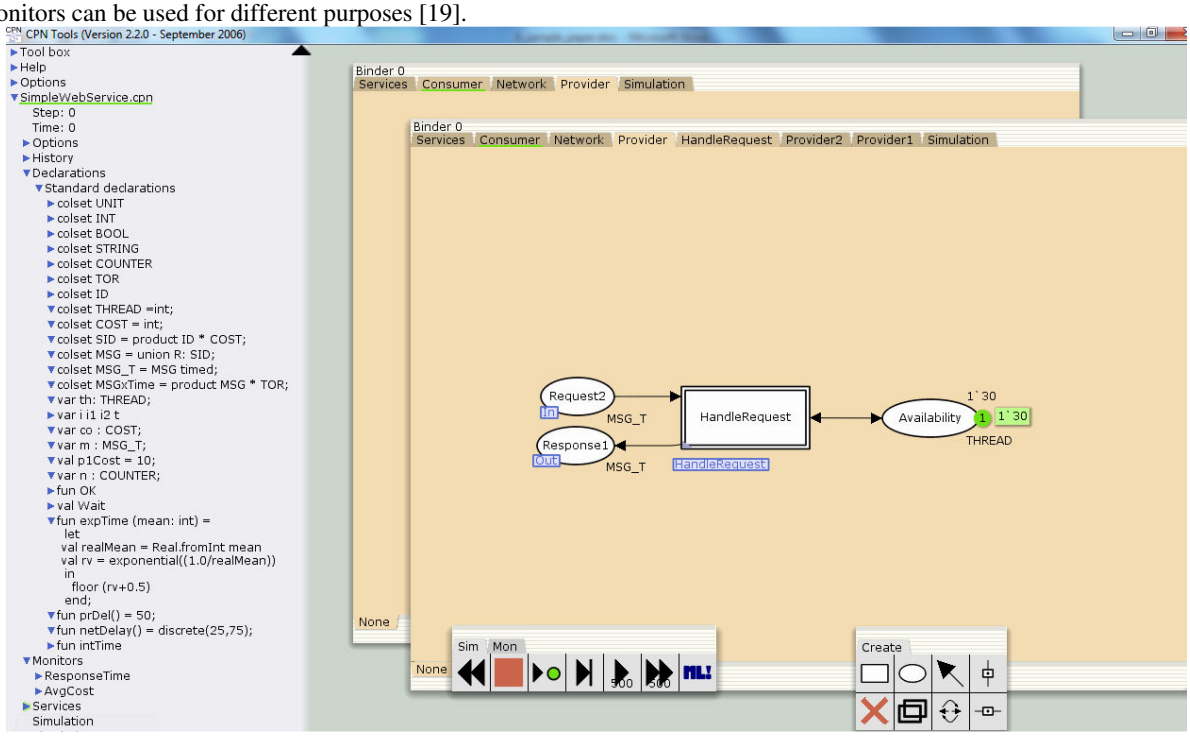

Figure 10 Screenshot of CPN Tools

in the Nodes ordered by pages entry. The predicate function determines when the monitor should collect data from model. The predicate function for RoundTripTime monitor looks like this:

fun predBindElem (Services'ReceiveResult (1,{i1,i2,s1,s2,t})) = true

The function returns true every time *ReceiveResult* occurs that it means a new response received by consumer. The observation function collects numerical data from the model whenever the predicate function for the same monitor is called and returned true. The observation function for measuring Response Time of messages subtracts the time of sending request t from the model time at which the transition occurs.

IntInf.toInt(time())-t

The initialization and stop functions are not used to collect data from markings. The data values that are returned by the observation function are used to calculate statistics. Simulation output is most often random and the statistics in a simulation performance report are unreliable. Confidence intervals are often used to evaluate the accuracy of performance measure estimates. Accurate confidence intervals can only be calculated for data values that are independent and identically distributed (IID) [21]. By running simulation replications on a net that contains data collector monitors, some of the statistics that are calculated at the end of each simulation are used to calculate more accurate and reliable statistics. For this reason, we run 5simulation replication that each one consists of 5000 steps.

The Throughput represents the number of completions of web service requests during an observation time interval. We use a data collector monitor on *ReceiveResult* transition that count the number of firing transition. Each of them shows that a request is responded completely. It is also necessary to define predicate, observation, initialization and stop functions for the Throughput monitor, even though these functions will not be used to collect data.

The Execution Cost represents the price that a service requester has to pay for invoking the service. We use a data collector monitor on *ReceiveResult* transition that sum the cost of each token when the transition fire. The average Execution Cost in each composition can be calculated at the end of a simulation by dividing the sum by the number of firing transition.

IX. RESULTS

The QoS metrics of models are examined for request arrival rates of 5, 10, 20, 40 and 50 requests/second. Ten independent simulations are executed for each web service composition that are examined. 95% confidence intervals are calculated for the in question. We will assume that the interarrival times between requests are exponentially distributed. This is not the most accurate model for request arrivals [26], but it proved to be sufficient for our purposes.

Simulation output is most often random and the statistics in a simulation performance report are unreliable. Confidence intervals are often used to evaluate the accuracy of performance measure estimates. Accurate confidence intervals can only be calculated for data values that are independent and identically distributed (IID) [21]. By running simulation replications on a net that contains data collector monitors, some of the statistics that are calculated at the end of each simulation are used to calculate more accurate and reliable statistics. For this reason, we run 5simulation replication that each one consists of 5000 steps.

Fig. 11 shows the plot of Throughput of service requests for two supposed model, described in Section VI. The x-axis represents the request/second and the y-axis represents the throughput as the number of completed response per second calculated from data collector monitor.

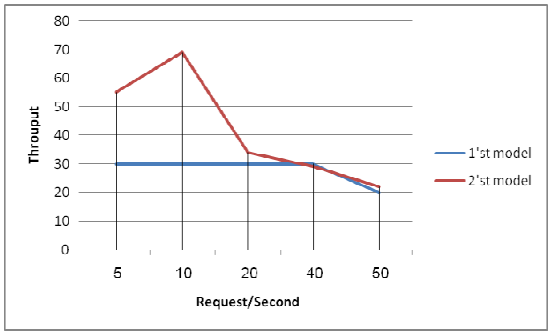

Figure 11: Throughput

Fig. 12 shows the Average Cost plot of two supposed model. The x-axis represents the request/second and the y-axis represents the average Cost calculated from data collector monitor.

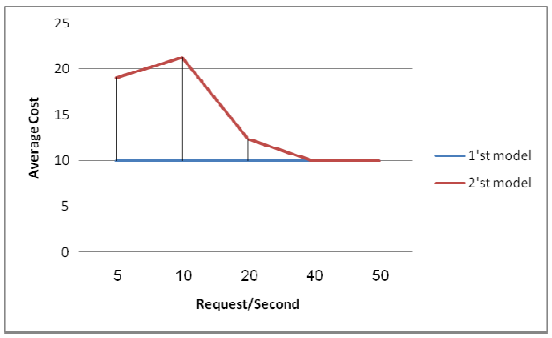

Figure 12: Average Execution Cost

Fig. 13 shows the Response Time plot of two supposed model. The x-axis represents the request/second and the y-axis represents the average Response Time calculated from data collector monitor.

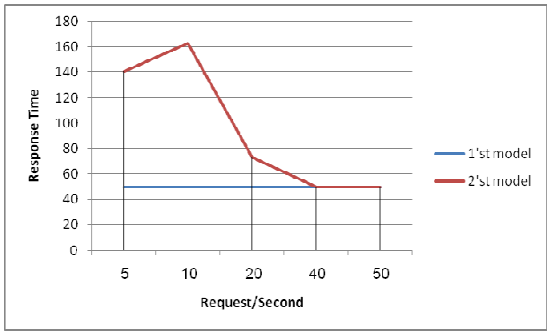

Figure 13: Average Response Time

.

The Throughput, Response Time and Average Cost are largest when the arrival rate is 10 request/second. As seen from these graphs, there is a sharp cut-off at inter-arrival value of 40 for two models. Increasing the arrival rate of requests more than 40 requests/second, two models work similarly for all our QoS metrics. When the request arrival rate is 10 requests/second, then the throughput is 60, i.e. more the requests completed and responded the providers. Even though in the arrival rate 10 requests/second we have the most Cost and Response Time.

X. CONCLUSION

In this paper, a simulation based approach was proposed for analyzing QoS metrics of web services composition. We used Colored Petri Net for modeling our alternative design. Also, we showed how we can use CPN Tools facilities in order to measure the QoS metrics of web services models. Petri nets were developed precisely for modeling such systems and scenarios. In fact, the graphical nature of the language gives a very visual representation of sequential and parallel composition, both asynchronous and synchronous communication, resource constraints, and mutual exclusion.[23] Colored Petri Nets (CPNs) are extensions of Petri Nets that allow modeling of both timed and untimed and support a powerful module mechanism that allows building of models in hierarchical manner. CPNs combine the strengths of ordinary Petri Nets with the strengths of a high-level programming language. Petri Nets provide the primitives for process interaction, while the programming language provides the primitives for the definition of data types and the anipulations of data values.

In fact we have successfully employed CPNs to model and analyze both the internal components and interactions as well as the external user/service request patterns and behaviors of an Enterprise Service Bus (ESB) developed defense applications [24, 25]. For analyzing QoS metrics, we used the time concept. The associated graphical software tool (CPN Tools) provides support for construction as well as analyzing CPN models. We used the simulation-based analysis supported by CPN Tools to analyze and compare our 2 simple web service composition.

In our work, we used CPNs with CPN Tools, to model both timed and untimed activities and to perform quality of service (QoS) analysis of web service composition.

XI. REFERENCES

[1] Newcomer E., Lomow G., 2004, Understanding SOA With Web Services, Addison Wesley.

[2] Erl T. , 2005, SOA; Concepts, Technology & Design, Prentice Hall PTR.

[3] Josuttis N., 2007, SOA in Practice, O'Reilly.

[4] M. Gudgin, M. Hadley, and T. Rogers. Web services addressing 1.0 - core. May 2002. http://www.w3.org/ TR/2006/REC-ws-addr-core-20060509/.

[5] Weerawarana S., Curbera F., Leymann F., Storey T. and Ferguson D., 2005, Web Services Platform Architecture: SOAP, WSDL, WS-Policy, SAddressing, WS-BPEL, WS-Reliable Messaging, and More, Prentice Hall PTR.

[6] Apte N., Mehta T., 2002, UDDI: Building Registry-Based Web Services Solution, Prentice Hall PTR.

[7] Sommerville I., 2007, Software Engineering, Eighth Edition, Addison Wesley.

[8] W. Reisig. Petri Nets, volume 4 of EATCS Monographs on Theoretical Computer Science. Spriger-Verlag, 1985.

[9] Rami Mounla,QoS-Aware Web Service Composition, NZCSRSC 2008, April 2008

[10] Hamadi R., Benatallah B., 2003, *A Petri Net-based* Model for Web Service Composition, Proceedings of the 14th Australasian database Conference (ADC'03), pp. 191-200.

[11] L. Wells, Performance Analysis using CPN Tools, Proceedings of the First International Conference on Performance Evaluation Methodologies and Tools 2006,ACM Press, 2006

[12] Srini Narayanan, Sheila McIlraith, Analysis and simulation of Web services Eleventh International World Wide Web Conference (WWW11), 2002.

[13] Peterson J., 1981, Petri Net Theory and the Modeling of Systems, Prentice Hall PTR.

[14] Reisig W., 1985, Petri Nets: An Introduction, Springer.

[15] Murata T., 1989, Petri Nets: Properties, Analysis and Applications, Proceedings of the IEEE, April 1989, Vol. 77(4), pp. 541-580.

[16] Standard ML of New Jersey. www.smlnj.org.

[17] J.D. Ullman. Elements of ML Programming. Prentice-Hall, 1998.f

[18] Y. Ma and C. Zhang. Quick convergence of genetic algorithm for QoS-driven Web service selection. Computer Networks. 2008. 52.

[19] L. Wells, Performance Analysis using Coloured Petri Nets. PhD Dissertation, Department of Computer Science, University of Aarhus, 2002.

[20] CPN Tools. Online: http://wiki.daimi.au.dk/cpntools/

[21] K. Jensen, L. M. Kristensen, and L. Wells. Coloured Petri Nets and CPN Tools for modelling and validation of concurrent systems. International Journal on Software Tools for Technology Transfer (STTT), 9(3–4):213–254, 2007.

[22] L. Wells, Performance Analysis using CPN Tools, Proceedings of the First International Conference on Performance Evaluation Methodologies and Tools 2006, ACM Press, 2006

[23] V. Gehlot, K. Edupuganti. Use of Colored Petri Nets to Model, Analyze, and Evaluate Service Composition and Orchestration. Proceedings of the 42nd Hawaii International Conference on System Sciences , 2009.

[24] V. Gehlot and A. Hayrapetyan. A CPN model of a SIPbased dynamic discovery protocol for webservices in a mobile environment. In Proc. 7th Workshop on Practical Use of Coloured Petri Nets (CPN□06), pages 197□216, 2006.

[25] V. Gehlot, T. Way, R. Beck, and P. DePasquale. Model driven development of a service oriented architecture (soa) using colored Petri nets. In Proc. First Workshop on Quality in Modeling, ACM/IEEE 9th International Conference on Model Driven Engineering Languages and Systems (QiM/MoDELS□06), 2006.

[26] M. Arlitt and C. Williamson. Internet web servers: Workload characterization and performance implications. IEEE Transactions on Networking, 5(5):631–645, 1997.