

A Petri net-based Approach to QoS-aware Configuration for Web Services*

PengCheng Xiong, YuShun Fan

Department of Automation

Tsinghua University

Beijing, China

xpc03@mails.tsinghua.edu.cn, fanyus@tsinghua.edu.cn

MengChu Zhou

Department of ECE

New Jersey Institute of Technology

Newark, NJ 07102-1982 USA

zhou@njit.edu

Abstract: *With the development of enterprise-wide and cross-enterprise application integration and interoperation towards web service, web service providers try to not only fulfill the functional requirements of web service users, but also satisfy their non-functional conditions in order to survive in the competitive market. A hot research topic is how to configure web services to meet their demand under a dynamic heterogeneous environment. This paper builds a web service configuration net based on Petri nets in order to exhibit web service configuration in a formal way. Then, an optimal algorithm is presented to help choose the best configuration with the highest quality of services (QoS) to meet users' non-functional requirements. Finally, the simulation results and related analysis prove the soundness and correctness of our model and algorithm.*

Keywords: Web service, Petri nets, optimization, modeling and analysis, simulation.

1. Introduction

Web Service [1] framework has evolved to become a promising technology for the integration of disparate software components using Internet protocols. Web service providers register web services through an UDDI registry. The web service that they intend to offer is defined by WSDL. Then, web service users/requestors discover the needed web services and send the requests via invocation interfaces. After the response from a web service provider, they invoke those services under SOAP. When any single web service fails to accomplish service requestor's multiple function requirements, multiple web services need to be dynamically configured together to form a web service composition. However, configuring

web services to meet special needs is not a trivial task. There are two main considerations.

One is functional requirement [2, 3]. In this respect, the configuration must ensure that all functions required by the requestor are provided by service elements. A dynamic configuration can be modeled as a functional decomposition of the overall requested function. As a formal digraph to present all of the dependency relationships, Service Dependency Graph (SDG) is popularly used to depict dependency relationship among web services [2, 3]. Although it provides a means for web service configuration descriptions in order to ensure functional interoperability among collaborating web services, SDG deals with only the functional aspect. It is not hard to imagine that service requestors will face with a large number of choices of service configurations that can provide the similar function.

Another consideration is non-functional [4-8], such as cost and QoS which are necessary for the evaluation, selection, and configuration of needed service compositions. Nevertheless, although there are many research papers [4], projects, such as METEOR-S [6], and middleware, such as GlueQoS [7] and SwinDeW [8] related to QoS aware web service selection, they do not consider that a service configuration is operating under an environment where the run-time performance is fluctuating and quality of the individual services are subject to change.

In this paper, we address the optimal configuration issues by concentrating on a) modeling and definition of the configuration and b) the optimal QoS searching algorithm under a varying environment.

2. Modeling web service configuration with Petri nets

* This work was supported by the National High Technology Research and Development (863) Program of China under Grant 2006AA04Z151 and the China National Science Foundation under Grant 60674080. The extended version of this paper is submitted to IEEE Transaction on System, Man and Cybernetics: Part A for review.

The web service provided may be dynamically reconfigured upon the service component updates, resource availability changes, or user requests. We model the configuration problem as Service Functional Dependency Configuration (SFDC). A web service provided may have multiple SFDCs. The functional dependency relationship of a configuration can be divided into two types, i.e., function combination and function selection. We describe these two basic types by applying AND and OR structures in Petri nets [9, 10] as shown in Fig.1. The following definition is used.

Definition 1: A Petri net is a 5-tuple $PN = (P, T, I, O, M)$ where:

1. $P = \{p_1, p_2, \dots, p_m\}$, $m > 0$, is a finite set of places.
2. $T = \{t_1, t_2, \dots, t_n\}$, $n > 0$, is a finite set of transitions, with $P \cup T \neq \emptyset$ and $P \cap T = \emptyset$;
3. $I: P \times T \rightarrow \{0, 1\}$, is an input function that defines the set of directed arcs from P to T ;
4. $O: P \times T \rightarrow \{0, 1\}$, is an output function that defines the set of directed arcs from T to P ;
5. $M: P \rightarrow N$, is a $m \times 1$ column vector whose i th component represents the number of tokens in the i th place, where $N = \{0, 1, 2, \dots\}$. An initial marking is denoted by M_0 .

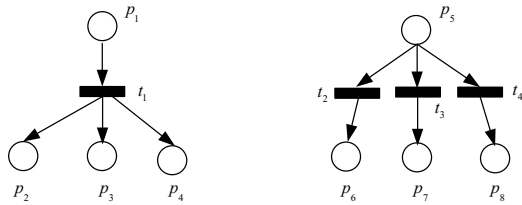


Figure 1. AND/OR structure in Petri nets

Post set of t is the set of output places of t , denoted by t^* . Pre set of t is the set of input places of t , denoted by *t . Post (Pre) set of p is the set of output (input) transitions of p , denoted by p^* and *p respectively.

Definition 2: Service set $S = WS \cup \{s_{dummy}\}$ where $WS = \{s_{1-z}\}$ is a finite set of web services, and s_{dummy} is dummy service.

Definition 3: Service Functional Dependency Configuration Net (S-net) : an acyclic Petri net PN is an S-net if:

1. $\forall s \in S, \exists p \in P$, s is mapped to a web service place p . If s is a dummy service, s is mapped to $p_{dummy} \in P$.

2. $\forall t \in T$, if $|{}^*t| > 1$ and $\forall p_1, p_2 \in {}^*t$, there is an AND relationship between p_1 and p_2 . $\forall t \in T$, if $|{}^*t| = 1$, ${}^*t = p$ and $|p^*| > 1$, there is an OR relationship among (p^*) .

3. Each transition has only one input arc and at least one output arc.

4. There is a place denoted by p' with no input arc, which corresponds to the service the user requests. $M_0(p') = 1$ and $M_0(p) = 0, \forall p \neq p'$.

5. A marking in an S-net is changed according to the following transition rule:

1) A transition $t \in T$ is said to be enabled if and only if $M(p) \geq I(p, t), \forall p \in P$ and we denote $M[t > .$ We denote $E(M)$ as an enabled transition set under M .

2) Firing t at M leads to M' , denoted as $M[t > M'$, where $M'(p) = M(p) + O(p, t) - I(p, t)$.

Definition 4: An SFDC, at time ζ , denoted as $C(p', \zeta)$ is a subset of P such that (a) $p' \in C(p', \zeta)$, and (b) $\forall p \in C(p', \zeta)$, if $p^* \neq \emptyset, \exists t \in p^*$ and $\forall p'' \in t^*, p'' \in C(p', \zeta)$.

3. Automatic web service configuration with optimal QoS

Each functionality of a service may have several QoS parameters and can be evaluated by measurements. Ranks the QoS parameters into runtime, transaction support, configuration management and cost, and security related ones [4]. Each is made up of several metrics and sub-metrics. For instance, higher values of throughput, reliability, robustness and flexibility are desired. Lower values of response time, latency and cost are also preferred. We suppose that a certain QoS parameter of a non-dummy web service place p at time ζ is $\psi(p, \zeta)$. In general, higher value of $\psi(p, \zeta)$ means higher quality. For example, $\psi(p, \zeta)$ can be throughput and denotes the number of completed service requests over a time period. To reflect the cost, $\psi(p, \zeta)$ is defined as the value of zero minus the price involved in requesting the service. If the web service represented by p is not available, $\psi(p, \zeta) = -\infty$.

Because the whole functionality of a service depends on its own functionality and that of its dependent sub-services, we assume that a QoS parameter for a $C(p', \zeta)$ is a function of the QoS parameter of all the non-dummy dependent services, i.e.,

$$QoS(C(p', \zeta)) = f(\psi(p, \zeta) \mid p \in C(p', \zeta) \setminus \{p_{dummy}\})$$

For example, the function to calculate the configuration cost can be simply summing up of all the non-dummy services'

$$QoS(C(p', \zeta)) = \sum_{p \in C(p', \zeta) \setminus \{p_{dummy}\}} \psi(p, \zeta)$$

Given p' and ζ , the algorithm to search for the best $C(p', \zeta)$ of S-net PN , i.e., an SFDC with the highest $QoS(C(p', \zeta))$ is as follows

OptimalConfiguration(p', ζ)

Initialization:

Get the $\psi(p, \zeta)$ for all the non-dummy web service places

Set all the transitions as unmarked

find=false

a marking stack $S = \{M_0\}$

OptimalQoS = $-\infty$ //The optimal QoS

OptimalS = \emptyset //The optimal marking stack

CStack = \emptyset //The configured web service place

stack

While ($S \neq \emptyset$)

Begin

Get the marking at the top of the stack $M = S.Top$

If $\exists t$ where $M[t >]$ and t has never been marked

with M

{

Mark t with M ;

For every $t' \in E(M)$, if $\bullet t' \neq \bullet t$, mark t' with

M ;

Push $\{\bullet t\}$ into *CStack*;

$M' = M[t >]$;

For every $p \in t^\bullet$, if $p \in CStack$

{

Delete the token in p and set $M'(p) = 0$;

}

Push M' into S ;

If (*find*==true) and $QoS(S) < OptimalQoS$

Pop(S) and *Pop(CStack)*;

}

Elseif $\exists t$ where $M[t >]$ but $\forall t$ has been marked

with M

{

Pop(S) and *Pop(CStack)*;

}

Else

{

If $QoS(S) > OptimalQoS$ //At this time the first or a better configuration is found.

{

find= true;

OptimalQoS = $QoS(S)$;

OptimalS = *Configuration(S)*;

}

Pop(S) and *Pop(CStack)*;

}

End Begin

End While

The QoS for a marking stack S can be calculated as follows:

QoS(Stack S)

{

Set temporary web service place set $P_T = \emptyset$

For every $p \in CStack$

Add p to P_T ;

Get the marking at the top of the stack $M = S.Top$

Find all $p \in P, M(p) > 0$

Add p to P_T ;

Return $f(\psi(p, \zeta) \mid p \in P_T \setminus \{p_{dummy}\})$

}

The configuration for a marking stack S can be calculated as follows:

Configuration(Stack S)

{

Set temporary SFDC $C(p', \zeta) = \emptyset$

For every $p \in CStack$

Add p to $C(p', \zeta)$

Get the marking at the top of the stack $M = S.Top$

Find all $p \in P, M(p) > 0$

Add p to $C(p', \zeta)$

Return $C(p', \zeta)$

}

Generally speaking, the QoS parameter of the configuration is deteriorating when more dependent web services are added to the configuration, e.g., compared with an individual web service, the integral of this web service and a dependent web service on it has worse reliability, longer latency and higher cost. Based on this regulation, as the algorithm proceeds, when the QoS parameter of the current configuration is already worse than the optimal one, we stop searching along this configuration path and retrieve to another one through *pop* and *push* operations. Hence the QoS parameter of $C(p', \zeta)$ found through the algorithm is increasing monotonically, and we can conclude that the algorithm is able to find the best $C(p', \zeta)$ with the highest

$QoS(C(p', \zeta))$. Only in the worst case, the algorithm has to search for the whole reachability tree.

Theorem 1[5]: If $|P \setminus \{p', p_{dummy}\}| = \eta$, the worst-case complexity of the above algorithm is approximately $O(e^\eta)$, where $e \approx 2.71828$ is the base of the natural logarithm.

Though the algorithm is proved to be of exponential time in the worst case, the complexity grows exponentially only with the number of places in OR structures (not all places) in the S-net and also depends on actual circumstance. On the other hand, the configuration step in our algorithm can be carried out concurrently, meaning that each sibling transition of t , i.e., $(\bullet t) \setminus \{t\}$ can fire with t concurrently. If multiprocessors are used, this property can speed up the algorithm. Moreover, the algorithm derives the best configuration result whose performance may well exceed what the user really expects. Hence, if considering users' actual expectation, the algorithm can terminate in a shorter time once it finds the first configuration meeting their expectation. The above points prove the applicability of our algorithm.

4. Case study and performance analysis

To evaluate algorithm *OptimalConfiguration*, we take an example and measure its configuration ability in user satisfaction rate through simulation. During the execution, at a given time period Y , we measure the number of the invoking times $Invoking(Y)$ and the number of times the user's request is satisfied $Satisfied(Y)$. Then the user satisfaction rate can be calculated as $Cr(Y) = Satisfied(Y) / Invoking(Y)$. We compare the performance of the proposed algorithm with two other typical approaches. The first method is *Fixed*, which selects the first feasible configuration and never changes. The other one is *Random*, which randomly chooses a configuration regardless of its performance.

In this section we take the example of manufacture execution service configuration as shown in Fig. 2. The manufacture execution service depends on the order management service to collect orders, workshop management service to schedule and monitor job progress, and process quality control service to track product quality. The job progress report service further relies on two kinds of monitoring services, i.e., the human inspection and live monitoring approaches. Both of the approaches are depend on a data collection service. The data collection service is a dummy service and shown in a dotted-line box, because the configuration should choose one and only one system to realize it.

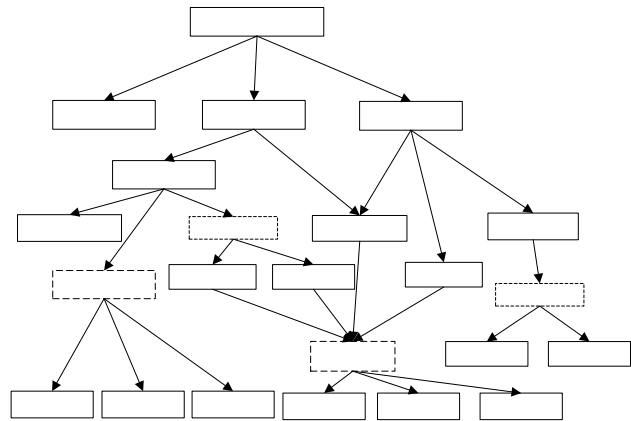


Figure 2. Web service dependency graph

The S-net corresponding to the web service dependency graph is shown in Fig. 3. For simplicity, here we take the cost of the web services as the QoS parameter. Suppose that in a highly varying environment, the cost of a web service changes conforming to a normal distribution. The web services that p' and p_{1-18} denote and their cost distribution parameters are shown in Tab. 1. We also assume that the user affordable price fluctuates in a normal distribution with a mean of 400 and a standard deviation of 20. The average user's request arrival rate is 0.2 per second and we approximately receive 600 web service requests over 3000 second period in the simulation environment.

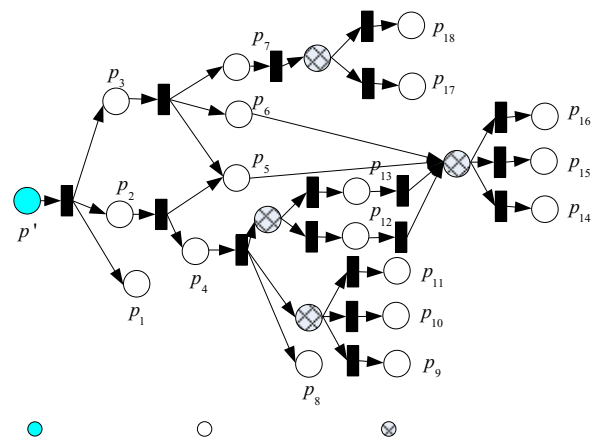


Figure 3. Web service dependency graph

We compare the performance of different algorithms in our simulation in Fig. 4 and Fig. 5. In the simulation, we choose the first feasible configuration as $\{p', p_{1-9}, p_{12}, p_{14}, p_{17}\}$ for *Fixed*. And it has the worst success rate because of its uniformly choosing one configuration. *Random* has better performance since it has more chances to obtain a better configuration. *OptimalConfiguration* algorithm keeps the best success

rate since it can accommodate its choice to the changing environment. Because the algorithm does not have to generate the whole reachability tree each time, it is also better than the exhausted enumeration method.

We then study the algorithm performance under a wider request range, i.e., the user affordable price has an increased mean from 350 to 500 and a standard deviation of 20. The simulation result is shown in Fig. 5. The curve for our *OptimalConfiguration* algorithm is always above the curves for *Random* and *Fixed* algorithms, which proves a better performance over the other two. Moreover, under a stringent user's request, e.g., 350, our *OptimalConfiguration* algorithm can still satisfy over 30% of the requests while the other two almost fail. When we gradually relax user's request to 500, the curve for *OptimalConfiguration* reaches 100% satisfaction rate rapidly, while the other two at a lower speed.

Note that, *Random* is not always better than *Fixed*. In the simulation environment, because the sum of normal distribution variables also conforms to a normal distribution, we can denote the cost obtained through the *Fixed* and *Random* algorithms as normal distributions $N(u_F, \sigma_F^2)$ and $N(u_R, \sigma_R^2)$ respectively, where $N(u, \sigma^2)$ denotes a normal distribution with a mean of u and a standard deviation of σ . From the cost distribution parameters in Tab. 1, it is easy to get $u_F > u_R$ and $\sigma_F \approx \sigma_R$ approximately. Suppose that the affordable price conforms to $N(u_U, \sigma_U^2)$. The success rate for the *Fixed* and *Random* algorithms can be calculated as $\Phi((u_U - u_F) / \sqrt{\sigma_F^2 + \sigma_U^2})$ and $\Phi((u_U - u_R) / \sqrt{\sigma_R^2 + \sigma_U^2})$ respectively, where $\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-t^2/2} dt$ is the Laplace function. Under the fixed affordable price mean of 400, we can calculate the satisfaction rate for *Fixed* as $\Phi(-0.8992) = 0.1841$, which is close to the simulation result in Fig. 4. Hence, if $u_F > u_R$ and $\sigma_F = \sigma_R$, *Random* has a higher user satisfaction rate. But if the first feasible configuration *Fixed* chooses the case that $u_F < u_R$, for instance, $\{p^1, p_{1-8}, p_{10}, p_{13}, p_{16}, p_{18}\}$, it has a higher user satisfaction rate than that of *Random*.

In summary, the above simulation results show that our algorithm is the best suitable for the highly varying environment, while delivering web services with high QoS upon a user's request to make the configuration decision.

Table 1. The web services and their cost distribution parameters.

Places	Web service name	Mean of cost	Standard deviation of cost
p^1	Manufacture Execution Service	10	3
p_1	Order Management Service	15	5
p_2	Workshop Management Service	20	5
p_3	Process Quality Control Service	12	4
p_4	Job Scheduling Service	40	10
p_5	Alarm and Warning Service	60	14
p_6	SPC Service	32	8
p_7	Product Tracking Service	30	9
p_8	Rolling Scheduling Service	38	10
p_9	PLC 1	30	9
p_{10}	PLC 2	22	7
p_{11}	PLC 3	26	11
p_{12}	Human Inspection Service	35	7
p_{13}	Live Monitoring Service	20	6
p_{14}	System 1	58	15
p_{15}	System 2	60	16
p_{16}	System 3	36	10
p_{17}	Bar Code Service	55	12
p_{18}	RFID Service	42	13

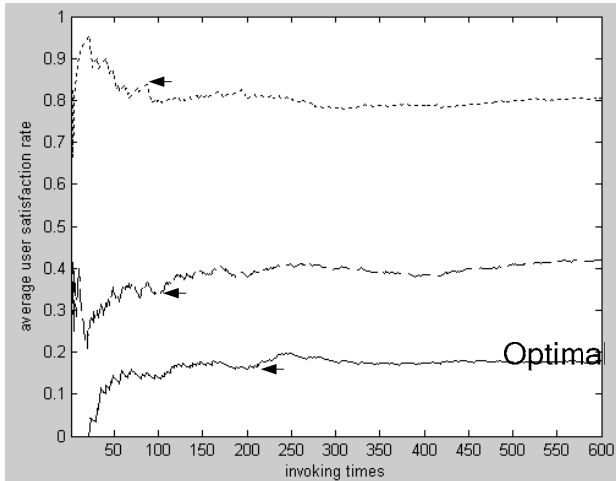


Figure 4. Simulation result under fixed affordable price mean of 400

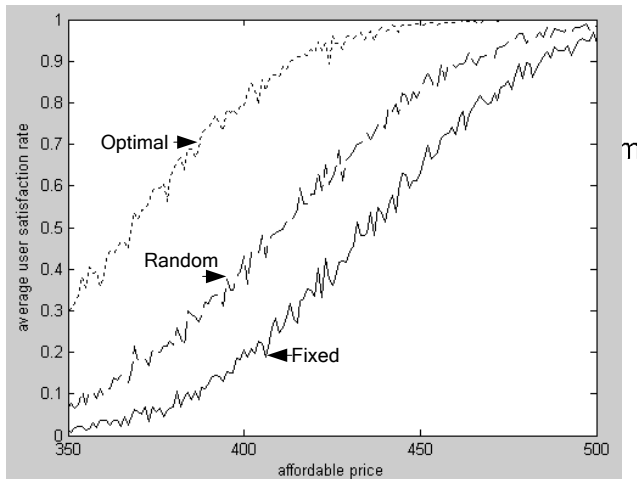


Figure 5. Simulation result under variable affordable price mean from 350 to 500

5. Conclusions

Since web service framework brings in a new revolution in traditional computing, web service configuration issues have been receiving more and more attentions. The importance of web service configuration is also incarnated in the research on other hot topics such as service fault management, fee payment as well as accounting management [5].

This paper proposes a systematic method to manage the existent web services to form a high performance configuration under a diverse and changeable environment. It presents a service functional dependency configuration net based on Petri nets for the web service

presentation and automatic configuration. Based on the configuration net, this paper presents an optimal algorithm able to return the optimal configuration with the best QoS parameter. Compared with a brute force exhaust algorithm, this algorithm often does not have to search for all the possible configurations in order to obtain the optimal one. Moreover, theoretical evidence and simulation results demonstrate that the configuration results our algorithm returns can achieve the highest user satisfaction rate.

References

- [1] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi and S. Weerawarana, "Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI", *IEEE Internet Computing*, Vol. 6, No. 2, Mar./Apr. 2002, pp. 86-93.
- [2] P. Hasselmayer, "Managing Dynamic Service Dependencies", *Proc. of the 12th Int'l Workshop on Distributed Systems: Operations and Management*, Nancy, France, Oct. 2001, pp.141-150.
- [3] A. Keller and G. Kar, "Dynamic Dependencies in Application Service Management", *Proc. of the International Conference on Parallel and Distributed Processing Techniques and Applications*, Las Vegas, 2002.
- [4] S. Ran, "A model for web services discovery with QoS", *SIGecom Exchanges*, Vol. 4, Iss. 1, Mar. 2003.
- [5] P.C. Xiong, Y. S. Fan and M.C. Zhou, "QoS-aware Web Service Configuration", *Submitted to IEEE Transaction on System, Man and Cybernetics, Part A*.
- [6] A. Sheth, J. Cardoso, J. Miller and K. Kochut, "QoS for Service-oriented Middleware", *Proc. of the Conference on Systems, Man and Cybernetics and Informatics*, Orlando, FL, July 2002.
- [7] E. Wohlstatter, S. Tai, T. Mikalsen, I. Rouvellou and P. Devanbu, "GlueQoS: middleware to sweeten quality-of-service policy interactions", *Proc. of the 26th International Conference on Software Engineering*, Edinburgh, United Kingdom, May 2004, pp. 189-199.
- [8] J. Yan, Y. Yang and G. K. Raikundalia, "SwinDeW-a p2p-based decentralized workflow management system", *IEEE Transactions on Systems, Man and Cybernetics, Part A*, Vol. 36, Iss. 5, Sept. 2006, pp. 922-935.
- [9] T. Murata, "Petri Nets: Properties, Analysis and Applications", *Proceedings of the IEEE*, 77(4), pp. 541-580, 1989.
- [10] M.C. Zhou and K. Venkatesh, *Modeling, Simulation and Control of Flexible Manufacturing Systems: A Petri Net Approach*, World Scientific, Singapore, 1998.