

QoS-aware Web Service Configuration

PengCheng Xiong, YuShun Fan, and MengChu Zhou, *Fellow, IEEE*

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 when the diversity of user requirements, distinction of service components' performance, and limitation of resources are considered. 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 service (QoS) to meet users' non-functional requirements. Finally, the simulation results and related analysis prove the soundness and correctness of our model and algorithm.

Index Terms—Modeling and analysis, optimization, Petri nets, simulation, web service

I. INTRODUCTION

WEB service [1]-[3] framework has evolved to become an important paradigm for distributed computing. Web service providers publish their web service and invocation interfaces they intend to offer in Web Services Description Language (WSDL) and register the web services to a common registration table located in Universal Description, Discovery, and Integration (UDDI). Then, application programs 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 Simple Object Access Protocol (SOAP) using asynchronous messaging or Remote Procedure Call (RPC) mode.

Web service providers must offer users with satisfied web services and establish a stable service connection with them. The calling mechanism is usually built on Service Level Agreements (SLA) [4]. Web service performance can be evaluated from two aspects. One is functional [5]. In this respect, the function a service can provide, under what circumstances the

function can be provided, should be completely matched with a user request. Another is non-functional, such as cost and quality of service (QoS) [6]-[7] which are necessary for the evaluation, selection, and configuration of services. Nowadays, UDDI based discovery of web services is based on the functional match only. It is not hard to imagine that there may be many web service configurations that can meet the functional requirement but carry different QoS attributes. Hence, the research on how to dynamically configure web services to meet users' non-functional requirement is receiving more and more attentions. However, the main challenge comes from the fact that the more flexibility and choices the provider offers, the more additional efforts need to be made to choose the best configuration. For example, according to the statistical data mentioned in [8], a group RPC service may have more than 200 candidate configurations. As a result, choosing a correct configuration with high performance needs high-level expertise and sufficient knowledge. In addition, a configurable service is running under a dynamic heterogeneous environment with different constraints. When a web service becomes unusable or unreliable, the other web services that depend on it should be altered in order to adapt to the new environment. Thus, to achieve run-time high performance enforcement, web service configuration should seek the collaboration among web services to make the configuration decision adapted to highly varying environment.

A web service configuration management problem is also related to other hot topics concerned with web service management. For example, fault management needs web service configuration information to track defects [9]. Accounting management pays and charges fee according to web service access and binding [10]. All these management activities need to know the current dependencies, discover their properties, and possibly perform the rebinding of services according to constraints. Therefore, it is necessary to develop a methodology for web service configuration considering not only the underlying service component and resource scheme, but also web service users' different requirements.

The rest of the paper is organized as follows: Section II briefly reviews the current research work on web service. Section III proposes a service function dependency configuration net based on Petri nets. Based on this net, the paper presents an optimal QoS configuration problem and an algorithm that can deliver the optimal configuration. Then its complexity and applicability are analyzed. Section IV evaluates the performance of our algorithm by comparing two other methods via simulation. Finally, Section V concludes the

Manuscript received September 13, 2006. This work was supported in part by the National High Technology Research and Development (863) Program of China under Grant 2006AA04Z151 and the China National Science Foundation under Grant 60674080.

P.C. Xiong and Y.S. Fan are with the Department of Automation, Tsinghua University, Beijing 100084, China (e-mail: xpc03@mails.tsinghua.edu.cn, fanyus@tsinghua.edu.cn)

M.C. Zhou is with Department of ECE, New Jersey Institute of Technology Newark, NJ 07102-1982 USA and is also with School of Electro-Mechanical Engineering, Xidian University, Xi'An, Shanxi 710071, China (e-mail: zhou@njit.edu).

paper.

II. RELATED WORKS

Web service framework brings in a new revolution in traditional computing. When a single web service fails to meet service user's multiple function needs, web services need to be dynamically configured together to form a web service composition. Service configuration and service composition complement each other in nature.

Service composition covers three distinct but overlapping viewpoints, i.e., behavioral interface, choreography and orchestration [11]. Behavioral interface captures a given individual service behavioral dependencies during the composition. Choreography deals with collaborative processes involving multiple services to achieve a common goal. Orchestration describes both control flow and data dependency of web services in a business environment which are often accomplished by the Business Process Execution Language for Web Services (BPEL) [12]. Since BPEL is specified informally, Petri nets [13] are used to perform formal verification of BPEL processes for efficient and effective composition of existing web services. Guided by the syntax of BPEL, Stahl realizes the semantic translation from BPEL processes to Petri nets [14]. Lohmann *et al.* present a framework based on Petri nets to formally analyze the interaction behavior of BPEL processes [15]. Zeng *et al.* propose a QoS-aware middleware for web services composition [16] and their modeling method guarantees the global optimization of linearly formulated QoS attributes. The above works are mainly process-oriented and provide the high level composition. However the assumption that every candidate web service can be associated with only one workflow task in [16] may contradict to the fact that a web service can support many workflow tasks and can be reused as many times as possible.

By offering the lower level business functions required by the orchestration, service configuration can deal with this problem in a function-oriented view. It is often done under Service Component Architecture (SCA) [17]. SCA builds on service encapsulation through the assembly of heterogeneous services. After the components that provide services and consume other services are implemented, they are assembled to build the business application through the wiring of services. Compared with process-oriented service composition, function-oriented service configuration enjoys more agility and flexibility. As an example, if a web service is configured by two services, these two services may be executed in sequence or parallel depending on their implementation.

SCA builds on service encapsulation through the assembly of heterogeneous services. After the components that provide services and consume other services are implemented, they are assembled to build the business application through the wiring of services. An SCA module is assembled by configuring and wiring together components, entry points and external services. Entry points are the representation of interfaces that are offered

for use by components outside the module. If components in the module depend on services provided outside the module, the services are represented as external services. SCA assembly operates at two levels, i.e., the assembly of components within a system and within a module. A dynamic configuration can be modeled as a functional assembly of the overall requested function. However, the SCA specification set being developed according to open service oriented architecture collaboration and most of the current research on SCA does not incorporate QoS [18].

As a dynamically formed digraph, Service Dependency Graph (SDG) [10] is widely used to depict dependency relationship within the selected web service sets. However, SDG is based on WSDL and deals only with the functional aspect. It is hard to support dynamic configuration of web service to deal with non-functional requirement. For example, Hasselmayer presents a dependency management architecture on web services and realizes the architecture with Jini connection technology [10]. Liang and Su propose an SDG based on AND/OR graph to discover web services [19]. The above studies focus on the functional aspect, without the consideration of users' non-functional requirements.

Moreover, although there are many research papers [6]-[8], projects, such as METEOR-S [20], and middleware, such as SwinDeW [3] and GlueQoS [21] 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. These studies motivate the research work presented in this paper.

III. DYNAMIC WEB SERVICE CONFIGURATION BASED ON PETRI NETS

The web service provided may be dynamically reconfigured upon the service component updates, resource availability changes, or user requests. Figure 1 shows an example of the web service dependency in a sales management service configuration. The sales management service depends on the order management service to collect orders, customer payment service to charge fee, and logistics service to deliver goods. The above three web services have an AND relationship with each other. The customer payment service further relies on two kinds of payment services, i.e., the remittance and bank account approaches. The approaches are shown in dotted line boxes and can be regarded as two dummy web services which have an OR relationship with each other. But both approaches have a functional dependency on a credit validation service to verify the customer's credit. The bank account service is also a dummy service and shown in a dotted-line box, because the configuration should choose one and only one bank service to realize it.

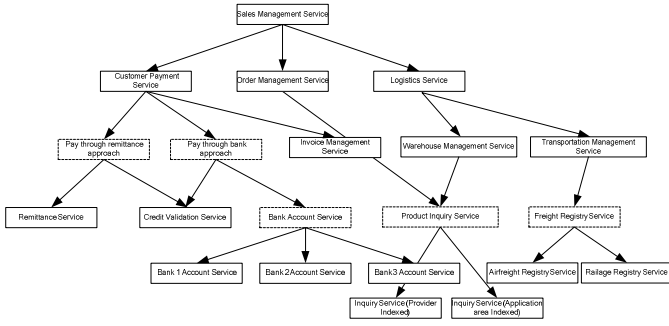


Fig. 1. Web service dependency graph

Therefore, a web service can choose to be functionally dependent on different sets of web services. We model this problem as Service Functional Dependency Configuration (SFDC). A web service provided may have multiple SFDCs. However, at one time, only one configuration can be selected for each web service. We transform the web service dependency graph into Petri nets by using two specific structures [22]-[24] in Fig. 2. i.e., transform function combination and function selection to AND and OR structures in Petri nets. After we add several related QoS parameters and measurements, we can derive the optimal web service configuration automatically. We assume that a reader is familiar with basic Petri nets [13]. The following definition is used.

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

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

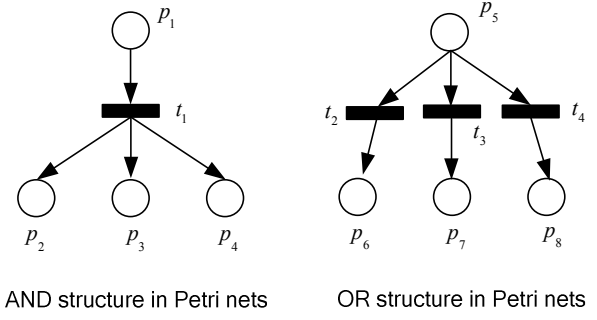


Fig. 2. AND and OR structure in Petri nets

Post set of t is the set of output places of t , denoted by t^\bullet . Preset of t is the set of input places of t , denoted by ${}^\bullet t$. Post (Pre) set of p is the set of output (input) transitions of p , denoted by p^\bullet and ${}^\bullet p$ respectively. In Fig. 2, $t_1^\bullet = \{p_{2-4}\}$, ${}^\bullet t_1 = \{p_1\}$, $p_5^\bullet = \{t_{2-4}\}$ and $(p_5^\bullet)^\bullet = t_2^\bullet \cup t_3^\bullet \cup t_4^\bullet = \{p_{6-8}\}$.

Definition 2: Service set $Se = WS \cup \{s_{dummy}\}$ where $WS = \{s_{1-z}\}$ is a finite set of web services, and s_{dummy} is dummy service which denotes a combination or selection of real web services.

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

- i. $\forall s \in Se, \exists p \in P$, s is mapped to a web service place p . If s is a dummy service, s is mapped to $p_{dummy} \in P$.
- ii. $\forall t \in T$, if $|t^\bullet| > 1$ and $\forall p_1, p_2 \in t^\bullet$, there is an AND relationship between p_1 and p_2 . $\forall t \in T$, if $|t^\bullet| = 1$, ${}^\bullet t = p$ and $|p^\bullet| > 1$, there is an OR relationship among $(p^\bullet)^\bullet$.
- iii. Each transition has only one input arc and at least one output arc.
- iv. There is a place denoted by p' with no input arc, which corresponds to the service the user requests. $M_0(p') = 1$ and $M_0(p) = 0$, $\forall p \neq p'$.
- v. A marking in an S-net is changed according to the following transition rule:

- a) 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 $E(M)$ as an enabled transition set under M .
- b) Firing t at M leads to M' , denoted as $M[t > M'$, where $M'(p) = 0$ if $(p, t) \in I$; $M'(p) = 1$ if $(p, t) \in O$ and $M'(p) = M(p)$ otherwise.

Note that the proposed S-net falls into the class of disassembly Petri nets [13].

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 $\forall p'' \in t^*$, $p'' \in C(p', \zeta)$.

For instance, Fig. 3 shows an S-net of the web service dependency graph in Fig. 1 and Tab. 1 shows the correspondent relationship between places and web services. Then one SFDC example is $C(p', \zeta) = \{p', p_{1-8}, p_{12}, p_{14}\}$ (for simplicity, we neglect dummy web services).

Each functionality of a service may have several QoS parameters and can be evaluated by measurements. The QoS parameters are sorted into runtime, transaction support, configuration management and cost, and security related ones [7]. 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., $QoS(C(p', \zeta)) = f(\psi(p, \zeta) | p \in C(p', \zeta) \setminus \{p_{dummy}\})$. For example, the function to calculate the configuration cost can be simply summing up of all the non-dummy services' $\psi(p, \zeta)$, i.e., $QoS(C(p', \zeta)) = f(\psi(p, \zeta) | p \in C(p', \zeta) \setminus \{p_{dummy}\})$

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

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 $t' \neq t$, mark t' with M ;

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

$M' = M[t >$;

For every $p \in t^*$, 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. So the QoS parameter of $C(p', \zeta)$ found through the algorithm is increasing monotonically. Thus we can conclude that when the algorithm terminates, it 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: 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.

Proof: We denote the configuration number as Ω . In the worst case, we have k dummy web services denoting k groups, each of which has an AND relationship with others. And in the i th group we have x_i web services, $i = 1, 2, \dots, k$, each of which has an OR relationship with others. Then, we have the

configuration number $\Omega \leq \prod_{i=1}^{i=k} x_i$ and $\sum_{i=1}^{i=k} x_i \leq \alpha\eta$, where

$\alpha = \text{Max}_{p \in P} (|\bullet p|)$. It is easy to prove that when

$x_1 = x_2 = \dots = x_k = \alpha\eta / k$, $\prod_{i=1}^{i=k} x_i$ reaches the maximum

$(\alpha\eta / k)^k$. In the worst case, we have to calculate and compare

$(\alpha\eta / k)^k$ times to achieve the best configuration. We take the

logarithm transformation of $(\alpha\eta / k)^k$ to obtain

$y(k) = k * \ln(\alpha\eta / k)$. Then $y'(k) = \ln(\alpha\eta) - \ln(k) - 1$, and

$y''(k) = -1/k$, where $y'(k)$ and $y''(k)$ denote the first and second derivative of function $y(k)$ respectively. Let $y'(k) = 0$, we have $k = \alpha\eta / e$. Since $y''(k) < 0$ (as $k > 0$), $y(k)$ reaches its maximum when $k = \alpha\eta / e$. Thus we have

$\Omega \leq \prod_{i=1}^{i=k} x_i \leq e^{\alpha\eta/e}$ and the complexity of the configuration step

is $O(e^{\alpha\eta/e})$ in the worst case. \square

Hence, the complexity of *OptimalConfiguration* algorithm in the worst case equals $O(e^\eta)$ approximately (considering relatively small α and e with η in a large S-net). 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. For example, though $\eta = 15$ and $\alpha = 2$ for the S-net in Fig. 3, there are only 16 optional configurations. 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 respectively. 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.

IV. 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 Υ , we measure the number of the invoking times $Invoking(\Upsilon)$ and the number of times the user's request is satisfied $Satisfied(\Upsilon)$. Then the user satisfaction rate is $Cr(\Upsilon) = Satisfied(\Upsilon) / Invoking(\Upsilon)$. 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.

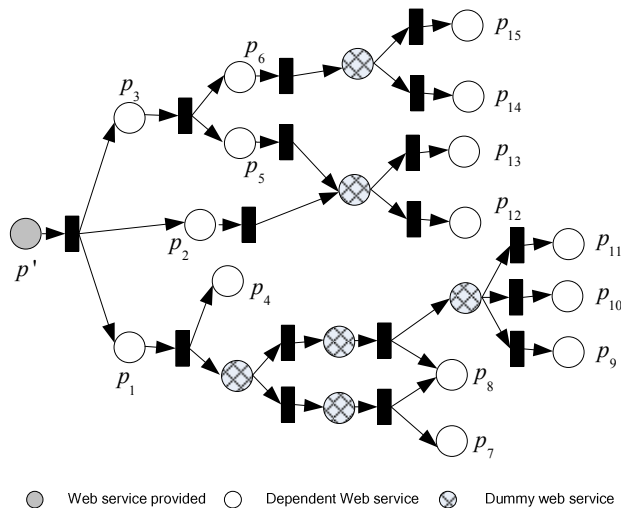


Fig. 3. An S-net for simulation

The practical example S-net is shown in Fig. 3, which corresponds to the web service dependency graph in Fig. 1. 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_{1-15} 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 390 and a standard deviation of 20. The average user's request arrival rate is 0.2 per second and we approximately receive 500 web service requests over 2500 second period in the simulation environment.

TABLE I

THE WEB SERVICES p' AND p_{1-15} DENOTE AND THEIR COST DISTRIBUTION PARAMETERS.

Places	Web service name	Mean of cost	Standard deviation of cost
p'	Sales Management Service	10	5
p_1	Customer Payment Service	15	5
p_2	Order Management Service	20	6
p_3	Logistics Service	50	10
p_4	Invoice Management Service	40	8
p_5	Warehouse Management Service	60	12
p_6	Transportation Management Service	35	8
p_7	Remittance Service	30	10
p_8	Credit Validation Service	60	10
p_9	Bank 1 Account Service	25	8
p_{10}	Bank 2 Account Service	22	8

p_{11}	Bank 3 Account Service	28	8
p_{12}	Inquiry Service (Provider Indexed)	20	7
p_{13}	Inquiry Service (Application area Indexed)	25	7
p_{14}	Airfreight Registry Service	40	7
p_{15}	Railage Registry Service	30	7

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 $\{p', p_{1-8}, p_{12}, p_{14}\}$ 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 often does not have to generate the whole reachability tree, it is also better than the exhaustive enumeration method as well.

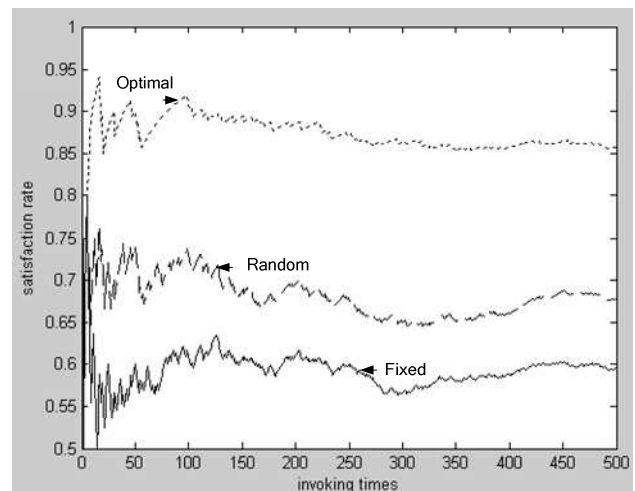


Fig. 4. Simulation result under fixed affordable price mean of 390

We then study the algorithm performance under a wider request range, i.e., the user affordable price has an increasing mean from 360 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. 360, our *OptimalConfiguration* algorithm can still satisfy over 60% 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.

However, *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 390, we can calculate the satisfaction rate for *Fixed* as $\Phi(0.2941) = 0.6141$, 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, $\{p'_1, p_{1-6}, p_8, p_{10}, p_{12}, p_{15}\}$, it has a higher user satisfaction rate than that of *Random*.

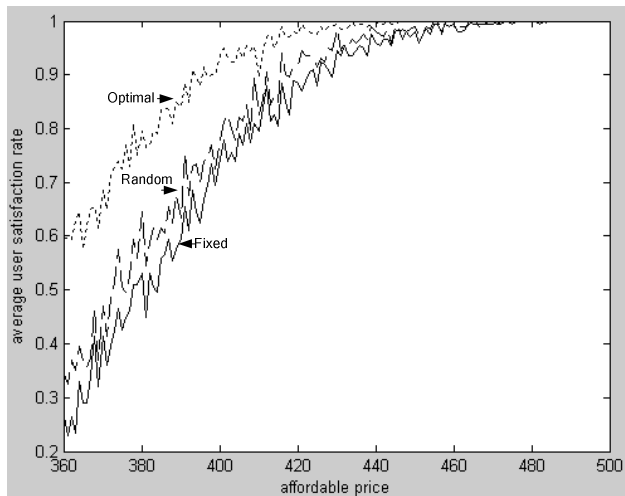


Fig. 5. Simulation result under variable affordable price mean from 360 to 500

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

V. CONCLUSIONS

Under diverse and changeable user requirements for the web services, and limitation of service components and resources, how to manage the existent web services to form a high performance configuration is becoming more and more important in order to best meet the non-functional requirement. Aiming at solving this problem, this paper proposes a systematic method.

Firstly, it proposes a service functional dependency configuration net based on Petri nets for the web service presentation and automatic configuration. Secondly, an optimal algorithm able to return the optimal configuration with the best QoS parameter is presented based on the configuration net. Compared with a brute force exhaustive search, this algorithm often does not have to search for all the possible configurations in order to obtain the optimal one. Note that both of the algorithm proposed by Zeng *et al.* for QoS-aware web services composition [16] and our algorithm are able to form a global highest performance configuration under diverse and changeable user requirements. However, compared with their algorithm which imposes an integer constraint on web service selection that adds prohibitively high computation cost for large-size problems, our algorithm not only takes web service reusability into consideration but also can be carried out concurrently on multiprocessors for better efficiency. Finally, theoretical evidence and simulation results shown afterwards prove that the configuration results our algorithm returns can achieve the highest user satisfaction rate and thus reflect the soundness and correctness of our work.

Moreover, the proposed algorithm could be easily integrated into existing web service environments as follows. First, according to the customized or application-specific requirement, discover all the web services. Second, by analyzing function decomposition and function selection on the service information, build a complete service functional dependency configuration net. Third, choose the QoS attributes for the whole configuration. Fourth, collect the QoS attribute value for each non-dummy web service. Finally, run the algorithm and the best configuration is obtained.

The future work should extend our algorithm to handle the case of user's requirements in multiple QoS aspects, e.g., a configuration with the lowest cost may not have the highest reliability. Finding some appropriate coordination mechanisms to balance the multiple requirements for a proper configuration is an important and interesting problem left for future exploration.

REFERENCES

- [1] "Web Service Activity", World Wide Web Consortium, www.w3.org/2002/ws/.
- [2] 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.
- [3] 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.
- [4] D.A. Menasce, "Mapping service-level agreements in distributed applications", *IEEE Internet Computing*, Vol. 8, Iss. 5, Sept.-Oct. 2004, pp. 100-102.
- [5] J. Cardoso and A. Sheth, "Semantic e-workflow composition", *Journal of Intelligent Information Systems*, Vol. 21, No.3, 2003, pp. 191-225.
- [6] D.A. Menasce, "QoS issues in Web services", *IEEE Internet Computing*, Vol. 6, Iss. 6, Nov/Dec. 2006, pp. 72-75.
- [7] S. Ran, "A model for web services discovery with QoS", *SIGecom Exchanges*, Vol. 4, Iss. 1, Mar. 2003, pp. 1-10.

- [8] C. Yi and K. Nahrstedt, "QoS-Aware Dependency Management for Component-Based Systems", *Proc. of the 10th IEEE International Symposium on High Performance Distributed Computing*, San Francisco, CA, USA, Aug. 2001, pp. 127-138.
- [9] 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, USA, 2002.
- [10] P. Hasselmayer, "Managing Dynamic Service Dependencies", *Proc. of the 12th International Workshop on Distributed Systems: Operations and Management 2001*, Nancy, France, Oct. 2001, pp. 141-150.
- [11] A. Barros, M. Dumas and P. Oaks, "Standards for Web Service Choreography and Orchestration: Status and Perspectives", *Lecture Notes in Computer Science*, Springer-Verlag, 2006, Vol. 3812, pp. 61-74.
- [12] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana, "Business Process Execution Language for Web Services", version 1.1, May 2003, <http://www-106.ibm.com/developerworks/webservices/library/ws-bpel>
- [13] M. C. Zhou and K. Venkatesh. Modeling, Simulation and Control of Flexible Manufacturing Systems: A Petri Net Approach. *World Scientific*, Singapore, 1998.
- [14] C. Stahl, "A Petri Net Semantics for BPEL", *Informatik-Berichte 188*, Humboldt-Universität zu Berlin, July 2005.
- [15] N. Lohmann, P. Massuthe, C. Stahl, and D. Weinberg, "Analyzing Interacting BPEL Processes", *Proc. of the 4th International Conference on Business Process Management*, Vienna, Austria, Sept. 2006, pp. 17-32.
- [16] L.Z. Zeng, B. Benatallah, A.H.H. Ngu, M. Dumas, J. Kalagnanam and H. Chang, "QoS-aware middleware for Web services composition", *IEEE Transactions on Software Engineering*, Vol. 30, Iss. 5, May 2004, pp. 311-327.
- [17] J.L. Fiadeiro, A.Lopes, L. Bocchi, "A Formal Approach to Service Component Architecture", *Proc. of the 3rd International Workshop on Web Services and Formal Methods*, Vienna, Austria, Sept. 2006, pp. 193-213.
- [18] Z.L. Zou and Z.H. Duan, "Building Business Processes or Assembling Service Components: Reuse Services with BPEL4WS and SCA", *Proc. of the 4th IEEE European Conference on Web Services*, Zurich, Switzerland, Dec. 2006, pp. 138-147.
- [19] Q.A. Liang and S.Y.W. Su, "AND/OR Graph and Search Algorithm for Discovering Composite Web Services", *International Journal of Web Services Research*, Vol. 2, No. 4, Oct.-Dec. 2005, pp. 48-67.
- [20] A. Sheth, J. Cardoso, J. Miller and K. Kochut, "QoS for Service-oriented Middleware", *Proc. of the Conference on Systemics, Cybernetics and Informatics*, Orlando, FL, July 2002, pp. 528-534.
- [21] E. Wohlstadt, 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.
- [22] J.Q. Li, Y.S. Fan and M.C. Zhou, "Performance modeling and analysis of workflow", *IEEE Transactions on Systems, Man and Cybernetics, Part A*, Vol. 34, Iss. 2, Mar. 2004, pp. 229-242.
- [23] R. Bouyekhf and A.E. Moudni, "On the analysis of some structural properties of Petri nets", *IEEE Transactions on Systems, Man and Cybernetics, Part A*. Vol. 35, Iss. 6, Nov. 2005, pp. 784-794.
- [24] J.H. Park, "A deadlock and livelock free protocol for decentralized Internet resource coallocation", *IEEE Transactions on Systems, Man and Cybernetics, Part A*. Vol. 34, Iss. 1, Jan. 2004, pp. 123-131.