A Petri-net and QoS Based Model for Automatic Web Service Composition

Bin Li, Yan Xu, Jun Wu, Junwu Zhu
School of Information Engineering, Yangzhou University
Yangzhou, China
e-mail: lb@yzu.edu.cn, changzhou213@gmail.com, j_wu@vip.sohu.net, jdkr@163.com

***Abstract*—Web services are widely used because of their features of interoperability, loose-coupled and platform independent. Web services composition is one of the most popular topics in service computing area. In this paper a method based on Petri net coverability problem and utility of Web services is proposed to handle automatic service composition. The former is for satisfying functional requirements of service composition while the latter corresponds to non-functional properties. For a set of given services, each service is translated into component of Petri net and the input/output places with same semantics are merged. According to user's input and desired output, the initial marking and target marking can be obtained. Then the coverability tree and coverability graph can be constructed. Next, the nodes on the coverability graph which can cover the target marking should be find out. If there are more than one coverability paths, we determine the smallest weight denoted as "SW" on each path. The weight is defined as the utility calculated according to the QoS. At last, services on the path with largest SW are selected as service executing sequence to reach the goal of automatic service composition.**

***Index Terms*—Petri net; Web services; automatic composition; coverability; QoS**

I. INTRODUCTION

As the rapid development of Internet, current computing environment has changed into large scale distributed scene. Due to the open environment and heterogeneous platform, different applications work together has become more and more complicated and difficult. There is no doubt that Web services bring new livingness to solve the problem. Web services have the feature of interoperability, loose-coupled and platform independent [1]. As a result, services are attached more and more importance.

On the one hand, different services run on different platform may be created in distinct way and implemented by different program languages. The user's requests need to compose reasonable according to application background. On the other hand, for the purpose of reuse, single Web service is impossible too complex. In other words, single service has limited function which can not satisfy the requirement of practice application. So it is also necessary to compose multiple services to accomplish more complicated tasks. A composite service is a service has more complex structure and more powerful function. Traditional method of service composition need man-made definition of interact process between Web services, and manual coding to complete the task. Both WS-BPEL [2] and WSCDL [3] provide support for service composition, in which described how to compose several basic services to more complicated services. However, neither of them can do their work automatically because they work on grammar level. What's more, both of them take no account of quality of services (QoS).

Automatic service composition requires the program or the Agent to automatically select and assemble suitable Web services to achieve the goal, while the users only need to give a task description with regular format on higher level. The description may consist of what task to complete and the quality requirements about the task. The method this paper proposed can properly satisfy this requirement.

The contribution of this paper on the one hand is to use the coverability problem of Petri net to judge whether the service composition satisfy the functional requirements. On the other hand, we extend the traditional coverability graph by adding weight on each edge as the accordance to select the best solution for composition. This paper focuses on the following work. Given a set of Web services, according to user's input and desired output, we judge whether there exists a subset of the services which can be composed to satisfy the user's requirement. If so, we will find out the subset. What's more, the executable order of the services also can be found. Our method is based on Petri net coverability problem. The given services are translated into Petri net, whose coverability tree and coverability graph can be soon constructed. At last the node on the coverability graph which can cover the target marking should be find out, then the services on the coverability path with largest SW means the executable service sequence to compose.

The rest of this paper is organized as follows. Some concepts about Petri net and service composition have been introduced in Section II. In Section III we give more details about our method. Two algorithms are given to depict how to compose services automatically. Subsequently, an example is used to show how our method performs. Related work is discussed in Section V. Finally, Section VI concludes.

II. CONCEPTS ABOUT PETRI NET AND WEB SERVICE COMPOSITION

Petri net theory comes from Carl Adam Petri's PhD thesis [4] and was greatly enriched in later decades. Because Petri net is fit to describe asynchronous and

Project number: 60903130, BK2007074, BK2009698, BK2009699.
Corresponding author: Bin Li, Yangzhou University, Yangzhou, China. Email: lb@yzu.edu.cn.

doi:10.4304/jsw.7.1.149-155

parallel computer system model, it is widely used in computer science today. This paper will use Petri net to solve the problem of automatic service composition, so it is necessary to introduce the related concepts briefly.

Definition 1 [5] (Petri net). A Petri net is a 5-tuple, $N = (S, T; F, W, M_0)$ where:

- S is a set of places and T is a set of transitions, $S \cap T = \varnothing$, $S \cup T \neq \varnothing$,
- $F \subseteq S \times T \cup T \times S$ is a set of flow relation,
- $dom(F) \cup cod(F) = S \cup T$, where

  $dom(F) = \{x \mid \exists y : (x, y) \in F\}$ is the domain of F,

  $cod(F) = \{y \mid \exists x : (x, y) \in F\}$ is the range of F,
- $W: F \rightarrow \{1,2,3,...\}$ is a weight function,
- $M_0: S \rightarrow \{0,1,2,3,...\}$ is the initial marking.

Definition 2 [6] (pre-set and post-set). Suppose $x \in X$ is arbitrary element of N,

- ${}^\bullet x = \{y \mid (y, x) \in F\}$ is called pre-set of x,
- $x^\bullet = \{z \mid (x, z) \in F\}$ is called post-set of x.

Definition 3 [6] (coverability). Suppose M and M' are two markings of Petri net $N = (S, T; F, W, M_0)$, if $\forall s \in S$: $M(s) \leq M'(s)$, we say M is covered by M', denoted by $M \leq M'$.

Definition 4 (coverability path). Suppose S is a node on coverability graph of Petri net N, S has marking M_s, if $M \leq M_s$, the path from root node of coverability graph to node S is called the coverability path of M.

Definition 5 (effective transition). Suppose M_g is the target marking of $N = (S, T; F, W, M_0)$, $||M_0||$ is the support set of M_0, $||M_g||$ is the support set of M_g, $\forall t \subseteq T$, when and only when

- $\exists s \in ||M_0||$, there exists a directed path from s to t,
- $\exists s' \in ||M_g||$, there exists a directed path from t to s',

then t is called effective transition.

There are multiple definitions for Web service composition in academe. This paper gives our definition as follows.

Definition 6 (Web service composition). Web service composition is to select a set of services from several available services. The selected services can form a new executable service which is more complex and can accept user's input to create the desired output according to specific construct method.

III. THE METHOD OF AUTOMATIC WEB SERVICE COMPOSITION

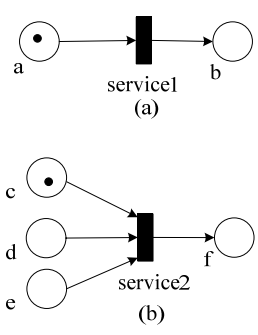

Figure 1. Service to Petri net

In the process of composing Web services, a set of services can be found in the register center such as UDDI [7]. The way of looking up service is based on key words match, so there are always multiple services can implement specific function. We consider QoS as the select standard. On the other hand, it is not clear whether exists a set of services among available services which can meet user's requirement exactly by composition. Petri net is useful tool to analysis this question.

A. From Web services to Petri net

Web services are functional entities with certain input and output. Each service has one or more input and output. We map every input and output of single service into places of Petri net. At the same time treat each service as a transition. Input places are the pre-set of transition, while output places are the post-set of transition. Services can be expressed as the form in Figure 1. Figure (a) means service1 has one input and one output. Figure (b) means service2 has three inputs and one output.

Users will provide necessary input, which can be looked upon tokens in the input places. For example, in Figure 1(a), one token in place *a* means user input the information about *a*. The reason for one token in place *c* in Figure 1(b) is the same.

In a large number of places, there will be several places with the same semantic meaning. All places with same semantics should be merged to one place with the tokens totally added. For instance in Figure 2(a), two services are listed. The service *bookInfoQuery* accepts *bookName* as input and output *bookInfo*. The service *bookDeal* accepts *bookName* and *quantity* as input and then output *bookDealList*. Here the "*bookName*" input in both services has the same semantics, so the two places are merged into one place, in which the number of tokens is sum of tokens before being merged. Here, the semantics of these places can be mapped from external ontology. Details are not discussed in this paper.

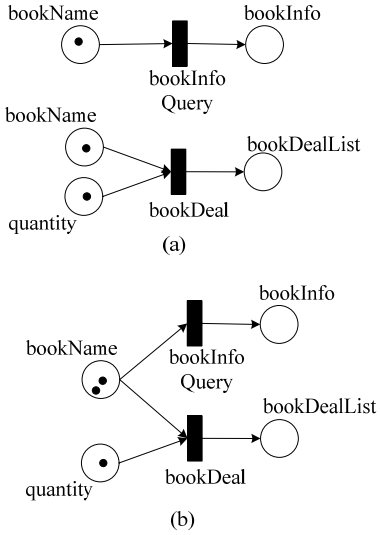

Figure 2. Merge the places with the same semantic meaning

After merging all the places with the same semantics, all the services organized as a Petri net. Of course, current Petri net may be connected or not. As we know transitions represent services, in terms of the definition of effective

transition in Definition 5, if a transition is not effective, there not exists causality between the transition and input or desired output. Obviously, the corresponding services will never be used. As a result, the services can be deleted from Petri net. This operation brings the benefit of reducing service select scope and improving the composition efficiency. According to the number of tokens in all the places initial marking can be obtained. Further more, the target marking M_g is got by setting desired output places with the value "1" and other places "0" instead.

B. The utility of services

In the register center, different services may have the same function with distinct qualities. If we choose bad service to compose, the overall quality of the composite service is impossible good. Therefore, we have to consider the quality about each member service. When given a set of service to compose to a specific one, if several solutions exist, we can determine the best solution by QoS.

In this paper we take QoS into account as following aspects.

- Response time. It denotes the time from requests send out to results back. The time consists of three parts. The following expression can calculate response time. $T_{response} = T_{sendRequest} + T_{execute} + T_{sendResult}$, where $T_{sendRequest}$ means the transmission time of the request send to the server, $T_{execute}$ is the execution time of the service and $T_{sendResult}$ denotes the duration of result transmitted to the user.
- Reliability. We use it to measure the probability of successful executing services. Here, success means the services can return accurate results. $R = s / n$, where s is the number of times the service successfully execute within a certain time, n is the number of invocations in the same time interval. Obviously, the higher reliability brings the better service quality.
- Usability. This property shows the probability of the services can be available on line since they published. The expression $U = T_{on} / (T_{on} + T_{off})$ can calculate the usability. T_{on} is the time of the service available on line while T_{off} figures out the failure time.
- Cost. It means the fee has to pay for the service. Whether the service is cheap depends on the provider.

Different QoS property has different measurement. For instance, response time measured by milliseconds or seconds, both reliability and usability are probability, and cost is scaled by currency. It's a problem how to turn these different measurements to the utility we need. Paper [8] introduced a method, which had two phases. In the first phase, namely scaling phase, QoS properties with different measurements were transformed to the values between [0, 1]. Then in weighting phase, the second phase, different QoS values multiplied different weights to get the overall utility of the service. The importance of the different QoS values depends on the user's preference. For example, some users take more importance on reliability and usability while some users prefer cheaper services.

C. Construct the coverability tree and coverability graph

In [5] a method to construct coverability tree is given. How to transform the coverability tree to coverability graph is also introduced. This paper extends the traditional coverability tree and graph by adding utilities of services on edges as weights of transitions. For net $N = (S, T; F, W, M_0)$, the algorithm to construct coverability tree $T(N)$ is listed as follows.

Algorithm 1

Input: the Petri net $N=(S, T; F, W, M_0)$, the value pairs of the transitions and the utilities $\{<t_1, u_1>, <t_2, u_2>, \dots, <t_n, u_n>\}$

Output: the coverability tree $T(N)$

(1) Label the initial marking M_0 as the root and tag it "new."

(2) While "new" markings exist, do the following:

(2.1) Select a new marking M.

(2.2) If M is identical to a marking on the path from the root to M, the tag M "old" and go to another new marking.

(2.3) If no transitions are enabled at M, tag M "dead-end."

(2.4) While there exist enabled transition at M, do the following for each enabled transition t at M:

(2.4.1) Obtain the marking M' that results from firing t at M.

(2.4.2) On the path from the root to M if there exists a marking M'' such that $M'(s) \geq M''(s)$ for each place s and $M' \neq M''$, that is, M'' is coverable, then replace $M'(s)$ by ω for each s such that $M'(s) > M''(s)$.

(2.4.3) Introduce M' as a node, draw an arc with label t from M to M', label the corresponding utility of t as the weight of this arc, and tag M' "new."

Using former algorithm we can get coverability tree $T(N)$, which can be translated to coverability graph $G(N)$ via establishing full mapping h: $T(N) \rightarrow G$, where [6]

- If x is a node on $T(N)$, then $h(x)$ is a node on G and $h(x)$ has the same label with M_x which is the label where x has on $T(N)$.
- If (x, y) is a directed arc on $T(N)$ with label of transition t and w as the weight, then $(h(x), h(y))$ is a directed arc on G with label of t and weight of w.
- x and y are the different nodes on $T(N)$, $h(x) = h(y)$ when and only when $M_x=M_y$ and both x and y on the same path on $T(N)$ from the root node r.

D. Automatic composing

The target marking M_g is determined by the desired output, i.e., each of the corresponding place has one token in it while there is no token in other places. Thereby, if there exists a node with marking M on the coverability graph, and $M_g \leq M$, then each desired output place has token under marking M. That is to say, subset of available services can be found out to satisfy user's requirement after composing. Here, on the coverability graph the node which can cover M_g may be not single. So we have to find out each coverability path for the corresponding coverability node. Then we determine the smallest weight,

denoted as "SW", on each path. The transitions on the coverability path with largest SW stand for the subset of available services can be composed. The executive order is from root to the coverability node on the path.

E. Algorithm for automatic composition

As we narrate previously, an algorithm for automatic Web services composition is concluded.

Algorithm 2

Input: a set of available services, user's input and requirement about QoS.

Output: executive service sequence or the information about composing failed.

(1) Translate each available service with the form of places and transitions, construct their binary relation.

(2) In terms of user's input, add a token to each corresponding place.

(3) Merge all the places with the same semantics to make the services compose to a Petri net. At the same time, the number of tokens in merged places is the sum of tokens before being merged.

(4) Check each transition whether it is an effective transition, if not, delete the transition. If its input places and output places become isolated, delete them as well as the tokens in, else decrease a token in input place on condition that there are tokens in before.

(5) Get the initial marking M_0 of the new Petri net.

(6) The target marking M_g is gained by setting desired output places with the value "1" and other places "0" instead.

(7) Compute the QoS value as the utility of each service.

(8) Construct the coverability tree and coverability graph.

(9) Look up all over the coverability graph

(9.1) If there exists a node S with marking M on the coverability graph, and $M_g \leq M$, then get the coverability path. If there are more than one node like this which bring more coverability paths, find out every smallest weight (SW) on each path. Compare with these SW values, transitions labeled on the path with maximal SW means the executive sequence of Web service.

(9.2) If there not exists any node whose marking can cover M_g, return composing failed.

IV. CASE STUDY

In this section a simple example is given to show how Algorithm 2 performed to compose services automatically.

Book query service
- t_0: key words → book name
- t_1: book name → book information

Deal service
- t_2: book name + address + quantity + zip code + phone number + recipients → total price + order form + delivery note

Pay service
- t_3: bank account + password + total price → payed
  QoS: higher reliability
- t_4: bank account + password + total price → payed
  QoS: lower reliability

Ship service
- t_5: order form + delivery note + payed → shipped
  QoS: longer response time and cheaper cost
- t_6: order form + delivery note + payed → shipped
  QoS: shorter response time and more expensive cost

Figure 3. Available services

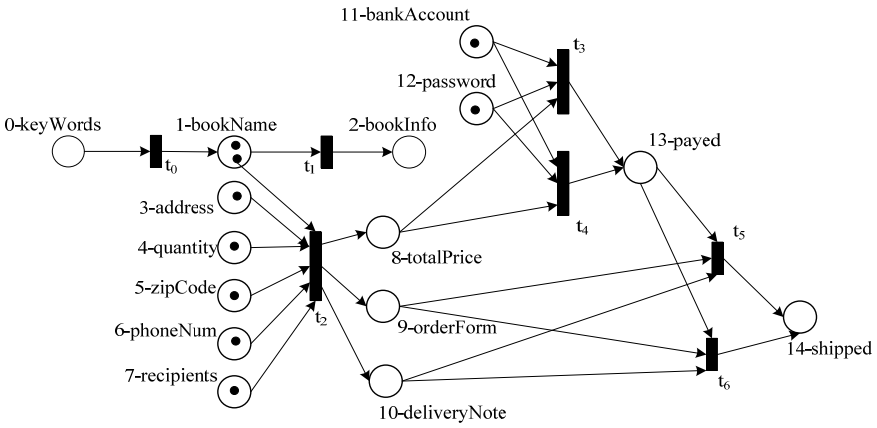

Figure 4. Translate services into Petri net

Supposing a user want to buy book on line. He or she can query the details about the books through *book query services*. When buying books, the user has to input the information such as book name and address. The buyer won't get books shipped from the seller until he or she pays on line. A few available services are listed in Figure 3. On the left of the symbol "→" are inputs of the services while the outputs are on the right side.

The user provides book name, address, quantity, zip code, phone number, recipients, bank account and password as input and relevant requirement on QoS. We assume the user prefers *pay service* with higher reliability

and *ship service* with shorter response time. The desired output is shipped.

As mentioned in previous section, first we represent every service with places and transitions. Then add tokens to places in terms of user input and merge places with same semantic meaning. The corresponding Petri net is shown in Figure 4.

According to the definition of effective transition, we find t_0 and t_1 are not effective transitions. So they can be deleted from the compositive Petri net to simplify the following operations. In terms of Algorithm 2, the places *keywords* and *bookInfo* can also be deleted. The number of tokens in *bookName* decrease 1. After simplifying the Petri net, it looks like in Figure 5.

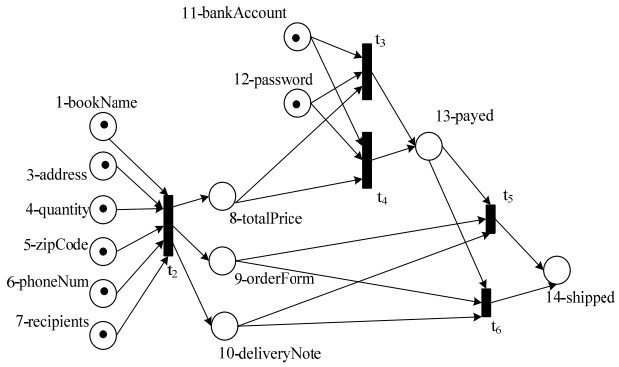

Figure 5. The simplified Petri net

The initial marking M_0= (1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0), and the target marking M_g= (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1).

The extended coverability graph is constructed as described in Algorithm 1 and Algorithm 2. It is shown in Figure 6. The weight of each transition keeps accordant with the QoS given in Figure 3.

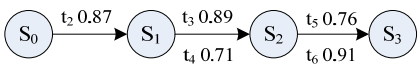

Figure 6. coverability graph

Here, S_0 is the same node with the root node on the coverability tree. Its marking is M_0= (1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0), node S_1 has marking (0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0), node S_2 has marking (0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0), node S_3 has marking (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1).

Only the marking of node S_3 can cover the target marking M_g, all the executable sequences are: $t_2 \rightarrow t_3 \rightarrow t_5$, $t_2 \rightarrow t_3 \rightarrow t_6$, $t_2 \rightarrow t_4 \rightarrow t_5$ and $t_2 \rightarrow t_4 \rightarrow t_6$. The smallest weights of them are 0.76, 0.87, 0.71 and 0.71. Obviously, $t_2 \rightarrow t_3 \rightarrow t_6$ has highest SW value 0.87. Thus, we find out the services can be composed to fit users' requirement, and achieve the goal of composing Web services automatically.

V. RELATED WORK

Many researchers in and out of home have made great study about services composition, mainly considered aspects as follows.

- Static service composition based on WS-BPEL

WS-BPEL mainly referred to Web service orchestration. A center node plays the role of commander who is responsible for coordinating other services by invoking them. Each member service is selected before running, as well as the flow is defined. If problems emergent when running, only compensate mechanism defined beforehand can work to handle the problems. Dynamic regulating is not supported. Therefore, the composition based on WS-BPEL is static. In paper [9] interoperation between Web services are described as business flow. The composition method is based on WS-BPEL, so it can't compose automatically.

- Dynamic composition based on semantic Web services

Semantic Web services relate to ontology. The goal of ontology [10] is to obtain knowledge about related domain, provide common comprehension for the domain knowledge and determine vocabulary in the domain. The knowledge provided by ontology enables the computer to deal with information automatically. Semantic Web services are described by ontology description language such as OWL-S [11]. User's preference and QoS requirements can also depict when compose services.

The composition of semantic Web services always combines with Multi-Agent System (MAS) [12] and Agent organization [13]. Some frameworks have already been proposed. Agent is initiative with reasoning capability, so it always used as a procurator for the service to coordinate the interactions among them. This composition method is suitable in dynamic environment. If the environment changed, the composite service can adapt itself, e.g. change the structure or the flow of composite service. The ALIVE project [14, 15] discussed the service how to self-adapt in dynamic environment.

- Automatic composition based on formal methods

Formal tools consist of automata, Petri net and process algebras.

In [16] the author study an abstract form of service composition where Web services are represented as nondeterministic communicating automata. The service composition problem consists, given a client service, a goal service and a community of available services, to determine whether there exists a mediator service able to communicate with the client and the services of the given community in such a way that their global behavior satisfies the client service request expressed as the given goal service.

In [17] a Petri net-based algebra for composing Web services is proposed. The formal semantics of the composition operators is expressed in terms of Petri net by providing a direct mapping from each operator to a Petri net construction. Thus, any service expressed using the algebra constructs can be translated into a Petri net representation.

In [18] an automatic service composition method is given. The services available are translated into a set of Horn clauses. User's input and output requirements are modeled as a set of facts and a goal statement in the Horn clauses respectively. Then Petri net is chosen to model the Horn clause set and T-invariant technique is used to determine the existence of composite services fulfilling the user's input/output requirements.

Paper [19] introduces a service-oriented software architecture called WS-NET which based on colored Petri net. WS-Net describes each web services component in three layers: interface net, interconnection net and interoperation net. It is executable, expressive and easy to use. However, transferring the WSDL specifications into the WS-Net specifications have to be manually.

In [20] the authors advocate the use of process algebras to describe, compose, and verify web services, with a particular focus on their interactions. To this aim, a case study is given in which they use CCS to specify and compose web services as processes.

VI. CONCLUSIONS AND FUTURE WORK

An automatic Web services composition method is proposed in this paper. Both functional and non-functional requirements are considered. For the functional aspect, Web services are translated into Petri net by merging the places with same semantics. Delete all the non-effective transitions and isolated places. Initial marking and target marking are determined in terms of user's input and desired output. Then construct the extended coverability graph of the Petri net to find out the node which can cover the target marking. If exists the node, that means the subset of given services can satisfy the functional needs. Otherwise, the composition fails.

We calculate utilities of each service through QoS properties and user's preference (e.g. which property is more important). When construct coverability graph the utilities are labeled onto related arcs as weights. If there are multiple coverability paths, find out every smallest weight of the services on each path. Choose the services on the path with maximal SW to compose. Non-functional requirement can also meet.

The method introduced in this paper can compose the Web services effectively. Nevertheless, this method didn't consider the potential restriction of executing order between services. We will take care of this problem next. In addition, if the composed service has runtime exceptions, a common solution is to replace the corresponding services. If there is no suitable service as alternative, the method proposed in this paper can also be useful to compose a service to replace. This is also our consideration in the future.

ACKNOWLEDGMENT

This paper is supported by the National Science Foundation of China under Grant No. 60903130, and the Natural Science Foundation of the Jiangsu Province of China under Grant No. BK2007074, BK2009698, BK2009699.

REFERENCES

[1] A. Tsalgatidou and T. Pilioura, "An overview of standards and related technology in Web services," Distributed and Parallel Databases, Vol. 12, Sep. 2002, pp. 135–162, doi: 10.1023/A:1016599017660.

[2] Web Services Business Process Execution Language Version 2.0, OASIS Standard, April 2007, Available at: http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html.

[3] Web Services Choreography Description Language Version 1.0, W3C Candidate Recommendation, November 2005, Available at: http://www.w3.org/TR/ws-cdl-10/.

[4] C. A. Petri, "Kommunikation mit automaten, " PhD thesis, Institut fur Instrumentelle Mathematik, Schriften des IIM 2, Bonn, Germany 1962.

[5] T. Murata. "Petri nets: properties, analysis and applications," Proc. IEEE, vol. 77, No. 4, Apr. 1989, pp. 541-580, doi: 10.1109/5.24143.

[6] C. Y. Yuan, "The principles and applications of Petri net," publishing house of electronics industry, Mar. 2005, (in Chinese).

[7] UDDI Spec TC Version 3.0.2, OASIS Standard, October 2004, Available at: http://uddi.org/pubs/uddi_v3.htm.

[8] L. Z. Zeng, B. Benatallah, A. H. H. Ngu, M. Dumas, J. Kalagnanam and H. Chang, "QoS-Aware Middleware for Web Services Composition," Software Engineering, IEEE Transactions on Volume 30, Issue 5, May 2004, pp:311 – 327, doi: 10.1109/TSE.2004.11.

[9] D. Mandell and S. McIlraith, "Adapting BPEL4WS for the Semantic Web: The Bottom-Up Approach to Web Service Interoperation," Proc. of the 2nd International Semantic Web Conference (ISWC 2003), LNCS 2870, 2003, pp. 227-241.

[10] N. F. Noy, "Semantic integration: a survey of ontology-based approaches," ACM SIGMOD Record Volume 33, Issue 4, Special section on semantic integration, December 2004, pp. 65-70.

[11] OWL-S: Semantic Markup for Web Services, W3C Member Submission, November 2004, Available at: http://www.w3.org/Submission/OWL-S/.

[12] M. Wooldridge, "An introduction to multiAgent systems," John Wiley and Sons, 2002.

[13] J. Ferber, O. Gutknecht and F. Michel, "From Agents to Organizations: An Organizational View of Multi-agent Systems," Agent-Oriented Software Engineering IV, Volume 2935/2003, Springer 2004, pp.443-459, doi: 10.1007/b95187.

[14] H. Aldewereld, L. Penserini, F. Dignum and V. Dignum. "Regulating organizations: The ALIVE approach," In Workshop on Regulations Modelling and Deployment (ReMoD-08), @CAISE'08, 2008, pp. 37-48.

[15] L. Penserini, H. Aldewereld, F. Dignum and V. Dignum, "Adaptivity within an organizational development framework," Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems, October 2008, pp. 477-478, doi: 10.1109/SASO.2008.13.

[16] P. Balbiani, F. Cheikh and G. Feuillade, "Composition of interactive Web services based on controller synthesis," Congress on Services - Part I, 2008. SERVICES '08, IEEE July 2008, pp: 521 – 528, doi: 10.1109/SERVICES-1.2008.11.

[17] R. Hamadi, B. Benatallah, "A Petri net-based model for Web service composition," Proc. Fourteenth Australasian Database Conference on Database Technologies, Vol. 17, Adelaide, Australia, 2003, pp.191-200.

[18] X. F. Tang, C. J. Jiang, Z. J. Ding and C. Chen, "A Petri net-based Semantic Web service automatic composition method," Journal of Software, Vol. 18, No.12, December 2007, pp. 2991-3000, doi: CNKI:SUN:RJXB.0.2007-12-006 (in Chinese with English abstract).

[19] J. Zhang, C. K. Chang, J. Y. Chung, S. W. Kim, "WS-Net: A Petri-net based specification model for Web services," Proc. IEEE International Conference on Web Services (ICWS 04), IEEE Press, Jun. 2004, pp. 420-427, doi: 10.1109/ICWS.2004.1314766.

[20] G. Salaun, L. Bordeaux and M. Schaerf, "Describing and reasoning on Web services using process algebra," Proc. IEEE International Conference on Web Services, July 2004, pp. 43-50, doi: 10.1109/ICWS.2004.1314722.

Bin Li was born in Yangzhou, Jiangsu Province, China, in 1965. He received the Ph.D. degree in computer application technology from Nanjing University of Aeronautics & Astronautics, Jiangsu, China in 2001.

Currently, he is a Professor of Yangzhou University and conducts research in the areas of service oriented computing, multi-agent system and artificial intelligence.

Yan Xu was born in 1984, M. S. candidate. Her main research interests include service computing oriented computing, software agent.

Jun Wu was born in Yangzhou, Jiangsu Province, China, in 1970. He received the Ph.D. degree in computer application technology from Southeast University, Jiangsu, China in 2005.

Currently, he is a Associate Professor of Yangzhou University and conducts research in the areas of service oriented computing, computer network and formal method.

Junwu Zhu was born in Yangzhou, Jiangsu Province, China, in 1972. He received the Ph.D. degree in computer application technology from Nanjing University of Aeronautics & Astronautics, Jiangsu, China in 2008.

Currently, he is an Associate Professor of Yangzhou University and conducts research in the areas of service oriented computing, Ontology and artificial intelligence.