International Journal of u- and e- Service, Science and Technology
Vol.7, No.2 (2014), pp.179-188
http://dx.doi.org/10.14257/ijunesst.2014.7.2.17

Towards the Automatic Generation of Petri nets for the OWL-S-based Complex Processes

Dongjin Yu and Zhiqing Liu

School of Computer, Hangzhou Dianzi University, Hangzhou 310018, China
yudj@hdu.edu.cn, 718529333@163.com

Abstract

The OWL-S-based composite process can describe the large-grained Web Service which contains complex relations and state transitions. However, it is very difficult to analyze the behavior of OWL-S composite process directly, without the aid of other formal models. On the other hand, Petri net is capable of modeling asynchronous, concurrent systems accurately. In order to reduce the complexity of analyzing OWL-S composite process, we introduce a method that converts OWL-S composite processes to ones described by Petri net. Based on the nine mapping rules presented in this paper, it translates the input, output and control structure of atomic processes into the corresponding elements in Petri net. We implement a tool named O2PJ, which converts OWL-S composite process into Petri net automatically based on our method, thereby reducing the complexity of analyzing OWL-S composite process.

Keywords: *large-grained Web service; OWL-S; Petri net; composite process; atomic process*

1. Introduction

In recent years, service-oriented computing (SOC) has become one of the most popular fields of software research. The core philosophy of SOC is to build loosely coupled collaborative software systems among the interaction of software components. SOC employs Web services as the basic components and a series of standardized protocols to interact among them. At the beginning, to build information system which is relative simple by compositing atomic services have been proved to be successful in practice. However, with the increasing complexity of business logic, more and more large-grained services which have rich feature and complex internal logic begin to appear in the Web service composition [1]. Large-grained service includes not only the syntax and semantics that static atomic services have, for example, inputs, outputs, messages, etc, but also includes the dynamic behavior, such as the internal control flow, data flow, interaction protocols, status changes and other dynamic attributes. In addition, large-grained services have relative complex and stable timing logic relationship [2]. Therefore, in order to describe, discover and compose the large-grained service, we should treat it in the different way from that we treat the atomic service.

At present, the studies on large-grained Web service have achieved many results. In general, we can use the description language, such as OWL-S and BPEL, to describe the state changes and business logic of the large-grained Web service accurately [3, 4]. In particular, Petri net is often employed in order to analysis behaviors of large-grained Web services [5-8]. For example, Tan et al propose the method to describe BPEL process using Petri net [3], whereas Ma et al describe how to use Petri net to describe the operational semantic of the OWL-S process, especially the composite process. However, the Petri nets in these approaches are generated manually. With the business logic of large-grained Web service becomes more and more complex, generating Petri nets of large-grained Web service manually is not quite feasible. Thus, according to the description of large-grained Web service to get the corresponding Petri net has important significance. In [9], Brogi et al present a method which focuses on the control flow and translates the OWL-S composite service into Petri net automatically. Meanwhile, Yu et al regard the pre-condition and effect as the core to describe the internal behavior of OWL-S composite process [10]. Vidal et al propose the mapping rules which are used to translate OWL-S process into Petri net in [11]. However they do not show how to make specific translating based on these rules. Unlike above studies, in this paper, we propose a novel method which regards the data flow as core to describe the internal logic of composite process and translates automatically the OWL-S composite process into the corresponding Petri net, which can clearly describe the dependencies among the atomic processes that make up of the OWL-S composite process.

The rest sections of this paper are organized as following. After Section II introduces the related concepts, Section III presents the mapping rules which translates the control construct of OWL-S composite process into Petri net. Section VI shows the method of translating the OWL-S composite process into Petri net in detail. After Section V introduces the tool called O2PJ which demonstrates the effectiveness of our approach, the final section concludes the paper and outlines the future work.

2. Related Concepts

2.1. Large-grained Web Service

At present, there is not a clear definition on the large-grained Web service in academia. However, many argue that the large-grained Web service contains multiple atomic services or smaller granularity services, and owns the complicated logical relationship and constraint conditions. The interaction of large-grained Web service may involve multiple interactive stateful processes, which adds much more complexity compared with the atomic Web service [12].

Definition 1 Large-grained Web service is composed of two or more Web services with smaller granularity and the simpler business logic. There are states and state changes during the execution of large-grained Web service.

2.2. OWL-S:Semantic Markup for Web Services

OWL-S is an ontology, within the OWL-based framework of the Semantic Web, for describing Semantic Web Services. It enables users and software agents to automatically discover, invoke, compose, and monitor Web resources offering services, under specified constraints [13]. The description files of an OWL-S is composed of the service profile, the process model and the grounding respectively [14, 15].

(1) Service profile: describing what the service could do.
(2) Process model: describing how to interact with the service.
(3) Grounding: assigning the details of agency-access service.

2.3. OWL-S Atomic Process

Definition 2 (OWL-S Atomic Process) An OWL-S Atomic Process can be represented with a triple of $AP = (I, O, A)$, in which I , O and A represent the input, output and action of the atomic process respectively [16].

2.4. OWL-S Composite Process

The Composite Process of an OWL-S described Web services mainly includes State sets, Action sets and Control structures of Sequence, Split-Join, Split, Choice, Repeat-While and Repeat-Until.

Definition 3 An OWL-S Composite Process can be represented with a quadruple of $CWS = (I, O, Ss, As, Cs)$, in which I , O , Ss , As and Cs represent the input, output, finite sets of state, finite sets of action and control structure of the composite process respectively.

2.5. Petri Net

A Petri net, also known as a place/transition net or P/T net, has a strong ability in describing the process of system or structure or the various structures between parts. It can be utilized to model and analysis the large-grained Web service [17, 18].

Definition 4 (Petri net) A Petri net can be represented with a triple of $PN = (P, T, F)$, in which:

(1) P represents the finite sets of place element expressed by circle.

(2) T represents the finite sets of transaction element expressed by rectangular.

(3) F represents the finite sets of acre element which represents the relationship between P and T. In other words, $F \in (P \times T) \cup (T \times P)$.

3. Mapping Control Structure of OWL-S Composite Process into Petri Net

The follows defines the mapping rules which translate the control structure of OWL-S composite process into Petri net.

Rule 1: The input and output of atomic process which constitute OWL-S composite process are mapped into the places of Petri net;

Rule 2: The atomic processes which constitute the OWL-S composite processes are mapped into the transactions of Petri net;

Rule 3: The relationships between the atomic process and its input, output are mapped into the acre of Petri net;

The control structure of the OWL-S composite process determines the execution order of atomic processes which it contains. These control structures are represented by <process:Sequence>, <process:Split-Join> and other labels, which use sub-labels of <list:first> and <list:rest> to control the execution order of atomic processes. Here, <list:first> and <list:rest> use the sub-label of <process:Perform> to include atomic processes. The rules of the mapping OWL-S control constructs into Petri net are illustrated in Figure 1, which are explained in detail as following.

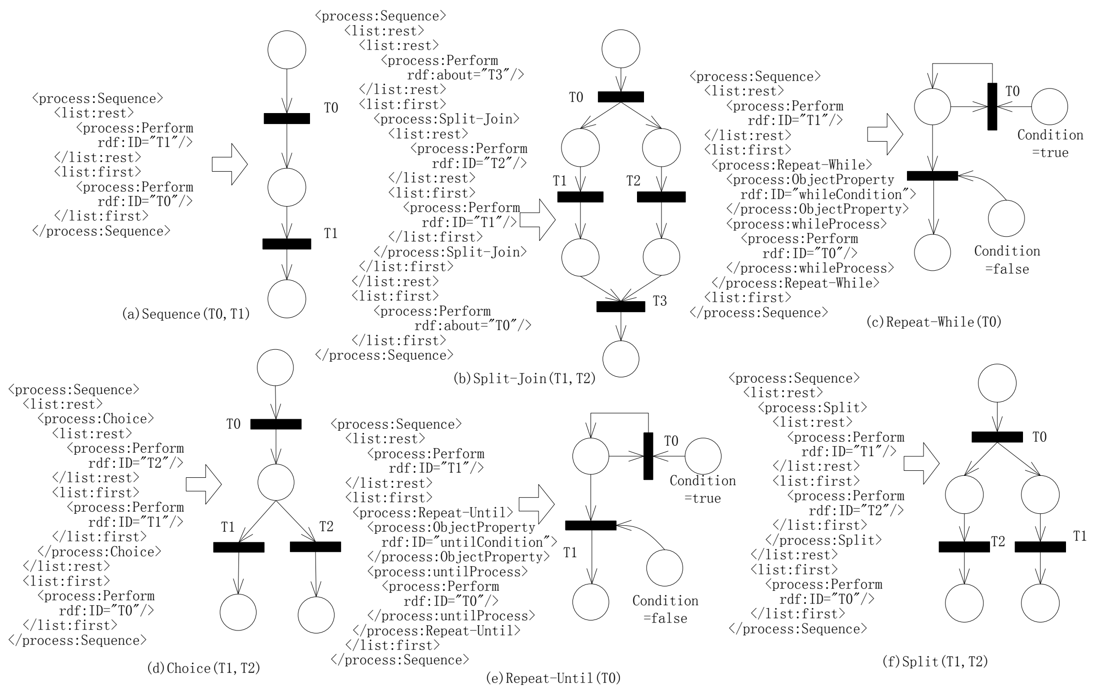
Rule 4-1 (Sequence mapping): The atomic process $T0$ represented by the label <process:sequence> and its sub-label <list: first> is translated into the transaction $T0$ of Petri net. The atomic process $T1$ represented by the label <process: Sequence> and its sub-label <list:rest> is translated into the transaction $T1$ of Petri net. $T1$ executes after $T0$, denoted by $T0.next = T1, T1.next = null$.

Rule 4-2 (Split-Join mapping): According to the rule 4-1, after the execution of atomic process $T0$, the atomic processes $T1$ and $T2$ which in split-join structure should be executed, $T1$ and $T2$ executed concurrently, after the execution of $T1$ and $T2$, atomic process $T3$ will execute. According to the rule 2, the atomic process T0 is translated into the transaction $T0$ of Petri net, the atomic processes $T1$ and $T2$ are translated into the transaction $T1$ and $T2$, the atomic process $T3$ is translated into the transaction $T3$, denoted by $T0.next = \{T1 \wedge T2\} T1.next = T3, T2.next = T3, T3.next = null$.

Rule 4-3 (Repeat-While mapping): After the execution of atomic process $T0$, if the condition is true, continue to execute atomic process $T0$, else atomic process $T1$ should be executed; The atomic process $T0$ represented by the label <process:Repeat-While> and its sub-label <process:whileProcess> is translated into the transaction $T0$ of Petri net, atomic process $T1$ is translated into the transaction $T1$, denoted by $if\ (condition = true)\ T0.next = T0;\ else\ T0.next = T1$.

Rule 4-4 (Choice mapping): According to the rule 4-1, after the execution of atomic process $T0$, the atomic process $T1$ or $T2$ which in Choice structure should be executed. According to rule 2, the atomic process $T1$ represented by the label <process:Choice> and its sub-label <list:first> is translated into the transaction $T1$ of Petri net. The atomic process $T2$ represented by the label <process:Choice> and its sub-label <list:rest> is translated into the transaction $T2$ of Petri net. The atomic process $T0$ represented by the label <process:Sequence> and its sub-label <list:rest> is translated into the transaction $T0$ of Petri net, denoted by $T0.next = \{T1 \vee T2\}, T1.next = null, T2.next = null$.

Rule 4-5 (Repeat-Until mapping): After the execution of atomic process $T0$, if the condition is true, continue to execute atomic process $T0$, else atomic process $T1$ should be executed; The atomic process $T0$ represented by the label <process:Repeat-Until> and its sub-label <process:untilProcess> is translated into the transaction $T0$ of Petri net, atomic process $T1$ is translated into the transaction $T1$; denoted by $if\ (condition = true)\ T0.next = T0;\ else\ T0.next = T1$.



```
<process:Sequence>
   <list:rest>
      <process:Perform
         rdf:ID="T1"/>
   </list:rest>
   <list:first>
      <process:Perform
         rdf:ID="T0"/>
   </list:first>
</process:Sequence>
```

```
<process:Sequence>
    <list:rest>
      <list:rest>
         <process:Perform
            rdf:about="T3"/>
      </list:rest>
      <list:first>
        <process:Split-Join>
          <list:rest>
             <process:Perform
                rdf:ID="T2"/>
          </list:rest>
          <list:first>
            <process:Perform
               rdf:ID="T1"/>
          </list:first>
        </process:Split-Join>
      </list:first>
    </list:rest>
    <list:first>
      <process:Perform
           rdf:about="T0"/>
    </list:first>
</process:Sequence>
```

```
<process:Sequence>
  <list:rest>
     <process:Perform
       rdf:ID="T1"/>
  </list:rest>
  <list:first>
   <process:Repeat-While>
     <process:ObjectProperty
       rdf:ID="whileCondition">
     </process:ObjectProperty>
     <process:whileProcess>
       <process:Perform
         rdf:ID="T0"/>
     </process:whileProcess>
   </process:Repeat-While>
  <list:first>
</process:Sequence>
```

```
<process:Sequence>
  <list:rest>
    <process:Choice>
      <list:rest>
        <process:Perform
           rdf:ID="T2"/>
      </list:rest>
      <list:first>
        <process:Perform
           rdf:ID="T1"/>
      </list:first>
    </process:Choice>
  </list:rest>
  <list:first>
    <process:Perform
       rdf:ID="T0"/>
  </list:first>
</process:Sequence>
```

```
<process:Sequence>
  <list:rest>
     <process:Perform
        rdf:ID="T1"/>
  </list:rest>
  <list:first>
   <process:Repeat-Until>
     <process:ObjectProperty
       rdf:ID="untilCondition">
     </process:ObjectProperty>
     <process:untilProcess>
       <process:Perform
          rdf:ID="T0"/>
     </process:untilProcess>
   </process:Repeat-Until>
  </list:first>
</process:Sequence>
```

```
<process:Sequence>
  <list:rest>
    <process:Split>
      <list:rest>
        <process:Perform
          rdf:ID="T1"/>
      </list:rest>
      <list:first>
        <process:Perform
          rdf:ID="T2"/>
      </list:first>
    </process:Split>
  </list:rest>
  <list:first>
    <process:Perform
      rdf:ID="T0"/>
  </list:first>
</process:Sequence>
```

Figure 1. Mapping OWL-S composite Processes into Petri Nets

Rule 4-6 (Split mapping): According to the rule 4-1, after the execution of atomic process $T0$,the atomic processes $T1$ and $T2$ which in split structure should be executed, $T1$ and $T2$ executed concurrently, the atomic process $T1$ and $T2$ represented by the label <process:Split> are translated into the transaction $T1$ and $T2$ of Petri net. The atomic process $T0$ represented by the label <process:Sequence> and its sub-label <list:first> is translated into the transaction $T0$ of Petri net; denoted by $T0.next = \{T1 \wedge T2\}, T1.next = null , T2.next = null$;

4. OWL-S Composite Process and Petri net Translating Method

This section discusses the method which translates the OWL-S composite processes into the Petri net automatically, as indicated by Table 1.

Table 1. The Algorithm for Translating OWL-S Composite Processes into the Petri Nets

```
Input
$CWS(I, O, Ss, As, Cs)$: the OWL-S composite process
Output
$PN(P, T, F)$: the corresponding description represented by Petri net
1. set $P(ws) = \phi$ , $T(ws) = \phi$ , $F(ws) = \phi$ ;  //initialize the Petri net
2. for each $A \in As$
3.   $A \to t$ ; //rule 2
4.   add $t$ to $T$ ;
5. for each $C \in Cs$
6.   switch( $C$ )
7.     case( $C$ equal Sequence):
```

```
8.        $t.next = t1$ ; //rule 4-1
9.        break;
10.   case( $C$ equal Split-Join):
11.        $t.next = t1 \wedge t2, t1.next = t3, t2.next = t3, t3.next = null$ ; //rule 4-2
12.        break;
13.   case( $C$ equal Repeat-While):
14.        $if\ (condition = true)\ t.next = t, else\ t.next = t1$ ; //rule 4-3
15.        break;
16.   case( $C$ equal Choice):
17.        $t.next = t1 \vee t2$ ; //rule 4-4
18.      break;
19.   case( $C$ equal Repeat-Until)
20.       $if\ (condition = true)\ t.next = t, else\ t.next = t1$ ; //rule 4-5
21.      break;
22.   case( $C$ equal Split)
23.       $t.next = t1 \wedge t2, t1.next = null, t2.next = null$ ; //rule 4-6
24.      break;
25. for each $S \in Ss$
26.   $S \rightarrow p$ ; //rule 1
27.    $p.from = t1$ ;   // $p$ is output of $t1$
28.   if $t1.next$ equal $t2$
29.      $p.to = t2$ ;   // $p$ is output of $t2$
30.          add $p$ to $P$ ;
31. for each $p1 \in P$
32.   for each $p2 \in P$
33.     //if $p1$ and $p2$ is the output of the same transaction, merge $p1$ and $p2$ ;
34.     if( $p1.from$ equal $p2.from$ )
35.        remove $p1$ ;
36. for each $t \in T$ and $p \in P$
37.   if( $p.to$ equal $t$ )
38.       $F[p,t] = 1$ ; // rule 3
39.   if( $p.from$ equal $t$ )
40.       $F[t,p] = 1$ ; // rule 3
41.return $PN$
```

Firstly, we parse the OWL-S description file to get the control structures and all the atomic process which constitute the composite process. According to Rule 2, we translate each atomic process into the transaction of Petri net and get the next atomic process of each atomic process based on rules from 4-1 to 4-6.

Secondly, we translate the input and output of atomic process into the places of Petri net according to Rule 1. If the subsequent atomic process of the atomic process t is $t1$, the output of the atomic process t is regarded as the input of the atomic process $t1$ In this way, we combine two places into one if necessary.

Finally, according to Rule 3, we translate the relationship of input and output of the atomic process into the relationship between places and transitions of Petri net. If the elements of place and transaction are linked, set the relationship to 1. Otherwise, set the relationship to 0.

5. Case Study

Based on the open source Petri net editor JSARP and the translating algorithm, we implemented a tool named O2PJ which is capable of translating composite process described by OWL-S into Petri-net-based one automatically. The follows presents an example based on O2PJ.

Suppose that a large-grained Web Service, *FindCheaperPrice*, is composed of four atomic services, *i.e.*, *FindBookNumber*, *FindBookStore1Price*, *FindBookStore2Price* and *ComparePrices*, described as follows.

(1) *FindBookNumber*: given the title of book output the ISBN number of this book;

(2) *FindBookStore1Price*: given the ISBN number of book output the price of this book in BookStore1;

(3) *FindBookStore2Price*: given the ISBN number of book, output the price of this book in BookStore2;

(4) *ComparePrices*: given two prices, output the cheaper one.

Table 2 shows the OWL-S description of large-grained Web Service *FindCheaperPrice*.

Table 2. The OWL-S Description of *FindCheaperBookPrice*

```
<process:CompositeProcess rdf:about="#FindCheaperBookPriceProcess">
......
 <process:Sequence>
  ......
   <list:rest>
     <list:rest>
       ......
       <list:first>
              <process:Perform rdf:about="#ComparePrices"/>
       </list:first>
     </list:rest>
     <list:first>
       <process:Split-Join>
         <list:rest>
            <list:first>
                   <process:Perform rdf:ID="FindBookStore1Price"/>
            </list:first>
         </list:rest>
         <list:first>
            <process:Perform rdf:ID="FindBookStore2Price"/>
         </list:first>
       </process:Split-Join>
     </list:first>
   </list:rest>
   <list:first>
     <process:Perform rdf:ID="FindBookNumber"/>
   </list:first>
 </process:Sequence>
</process:CompositeProcess>
```

The above OWL-S description file can be imported into O2PJ to get its corresponding Petri net. As presented in Figure 2, the left bottom corner shows the structure of composite process which is describe by the imported OWL-S description file, whereas the drawing area on the right shows the corresponding generated Petri net of the large-grained service.

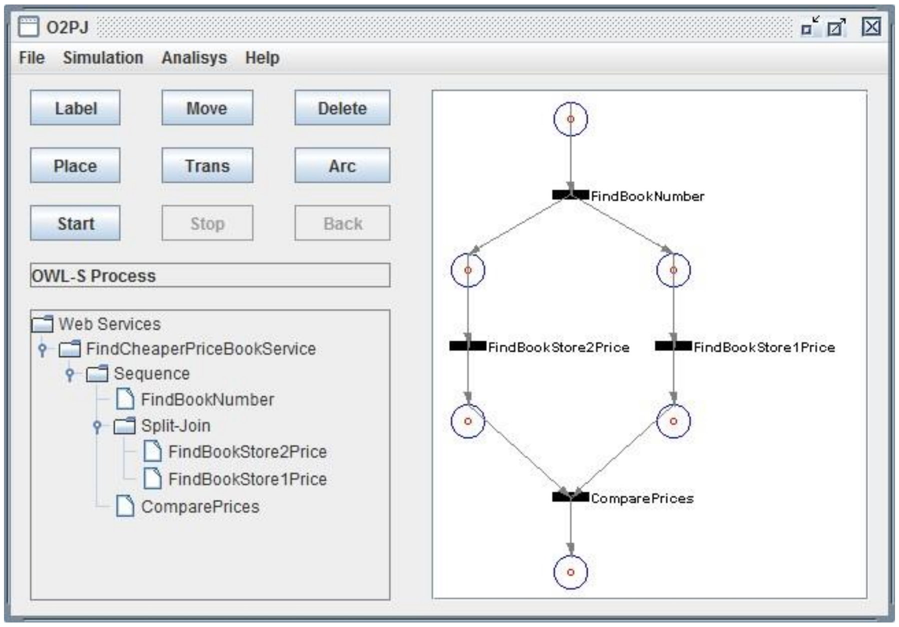


Figure 2. The Example that Translates the Composite Process based on O2PJ

6. Conclusions

This paper presents a method which translates OWL-S composite process into Petri net automatically. Compared with the traditional work such as [9] and [10], the method in this paper focuses on the data flow of large-grained Web services. It analyzes the relationship among the individual atomic processes of the composite process represented by OWL-S, and translates the atomic process, input, output and control construct to the corresponding elements of Petri net. The translated model represented by Petri net can accurately describe the internal logic of OWL-S composite process and the dependencies of atomic processes which constitute OWL-S composite processes. The experiment confirmed the effectiveness of this method.

In the future, we will focus on the improvement of O2PJ. Current version of O2PJ does not use PNML to describe Petri net, causing some other tools impossible to analyze the Petri-net-based models generated by O2PJ. Therefore, we will update O2PJ to add the support of PNML in the future.

Acknowledgements

This work is supported by National Science Foundation of China(Grant No.61100043), Key scientific and technological project of Zhejiang Province (Grant No.2012C11026-3), and National Science Foundation of Zhejiang Province(Grant No.LY12F02003).

References

[1] L. Xitong and F. Yushun, “Analyzing Compatibility and Similarity of Web Service Processes”, Chinese Journal Of Computers, vol. 32, no. 12, **(2009)**, pp. 2429-2437.
[2] M. E. Cambronero, G. Díaz, E. Martínez and L. Tobarra, “WST: a tool supporting timed composite Web Services Model transformation”, Simulation, vol. 88, no. 3, **(2012)**, pp. 349-364.
[3] W. Tan, Y. Fan and M. C. Zhou, “A petri net-based method for compatibility analysis and composition of web services in business process execution language”, IEEE Transactions on Automation Science and Engineering, vol.6, no.1, **(2009)**, pp. 94-106.
[4] N. Milanovic and M. Malek, “Current solutions for web service composition”. IEEE Internet Computing, vol. 8, no. 3, **(2004)**, pp. 51-59.
[5] R. Hamadi and B. Benatallah. “A Petri net-based model for web service composition”, Proceedings of the 14th Australasian database conference, **(2003)** February; Adelaide, Australian.
[6] D. Wenli, Y. Hang, Z. Yubing, “Testing bpel-based web service composition using high-level petri nets”, The 10th IEEE International Enterprise Distributed Object Computing Conference, **(2006)** October 16-20; Hong Kong, China.
[7] Y. Tang, L. Chen, H. Kaitao, N. Jing, “SRN: an extended Petri-net-based workflow model for Web service composition”, The 2004 IEEE International Conference on Web Services, **(2004)** July 6-9; San Diego, California, USA.
[8] J. C. Vidal, M. Lama and A. Bugarín, “Toward the use of Petri nets for the formalization of OWL-S choreographies”. Knowledge and information systems, vol. 32, no. 3, **(2012)**, pp. 629-665.
[9] A. Brogi, S. Corfini and S. Iardella, “From OWL-S descriptions to Petri nets”, Service-Oriented Computing-ICSOC 2007 Workshops, **(2009)** August 12; Vienna, Austriaz
[10] Y. Bo, L. Duluo, “Verifying web services of OWL-S with Petri net”, The 5th International Conference on Computer Science and Education, **(2010)** August 24-27; Hefei, China.
[11] J. C. Vidal, M. Lama and A. Bugarın. “Petri net semantics for OWL-S service choreography” Proceedings of the 4th international workshop on formal aspects of business processes and web services. **(2007)**, September 28-29; Brisbane, Australia.
[12] H. Haitao, L. Gang, H. Yanbo, “An Approach to Business-User-Oriented Larger-Granularity Service Composition”, Chinese Journal Of Computers, vol.28, no.4, **(2005)**, pp. 694-703.
[13] M. Klusch, B. Fries, K. Sycara. “OWLS-MX: A hybrid Semantic Web service matchmaker for OWL-S services”, Web Semantics: Science, Services and Agents on the World Wide Web, vol. 7, no. 2, **(2009)**, pp. 121-133.
[14] C. Guorong, T. Qqingping, W. Hao. “Model-Based Protocol Behavior Adaptation of Web Services in OWL-S”, Proceedings of the 2012 IEEE Asia-Pacific Services Computing Conference, **(2012)** December 6-8; Guilin, China.
[15] D. Martin, M. Burstein, D. Mcdermott, S. Mcllraith, M. Paolucci, K. Sycara, D. L. McGuinness, E. Sirin, N. Srinivasan, “Bringing semantics to web services with OWL-S”. World Wide Web, vol. 10, no. 3, **(2007)**, pp. 243-277.
[16] D. Martin, M. Burstein and J. Hobbs, “OWL-S: Semantic markup for web services”. W3C member submission, vol. 22, **(2004)**, pp. 2007-04.
[17] X. Pengcheng, F. Yushun S and Z. Mengchu. “A Petri net approach to analysis and composition of web services”. IEEE Transactions on Systems, Man and Cybernetics, vol. 40, no. 2, **(2010)**, pp. 376-387.
[18] N. Milanovic, M. Malek. “Current solutions for web service composition”, Internet Computing, vol.8 , no. 6, **(2004)**, pp. 51-59.

Authors

Dongjin Yu is currently a professor at Hangzhou Dianzi University, China. He re-ceived his BS and MS in Computer Applications from Zhejiang University in China, and PhD in Management from Zhejiang Gongshang University in China. His current research efforts include service computing, program comprehension and cloud computing. He is especially interested in the novel approaches to constructing large enterprise information systems effectively and efficiently by emerging advanced information technologies. The concern of his research closely relates with real applications of e-government and e-business. He has led a number of government funded projects, including the Rapid Application Development Framework, the OLAP Middleware, and the Cloud-based platform. He is the director of Insitute of Cloud Computing and Big Data and vice director of Institute of Intelligent and Software Technology of Hangzhou Dianzi University. He is also a member of ACM and IEEE, and a senior member of China Computer Federation (CCF).

Zhiqing Liu received his master degrees in computer science from Hangzhou Dianzi University in China. He has conducted a number of government funded projects related with Service Computing. His main research area is middleware technology and Software architecture.