Test Case Generation of Web Service Composition based on CP-nets

Yuying Wang
Science School, Xi'an University of Architecture and Technology, Xi'an, Shaanxi, China
Email: yinyin632@sina.com

Ning Yang
The School of Automation, Northwestern Polytechnical University, Xi'an, Shaanxi, China
Email: ningyang@nwpu.edu.cn

***Abstract*—Web service composition is an error prone task. Based on CP-nets (colored petri net) models, an approach of test case generation is proposed for web service compositions coded in BPEL. In this approach the semantic of BPEL concurrence and some special features are well dealt. Firstly, BPEL processes of a web service composition is translated into CP-nets models, then depth-first traversal works on the models immediately, and results in some sequence test paths. Secondly, after these sequence test paths merged into program executable units (PEU for short), the constraint set of these units is solved and filtered and formed into test cases. Finally, an application of the approach is illustrated with an example, which more efficiency shown with 7 test units less than 9 test paths appeared in a reference for same example, 3 test cases far less than formal works.**

***Index Terms*—web service composition, test case generation, CP-nets, state space explosion, BPEL.**

I. Introduction

Web service composition is an error prone task in which service candidates interact complexly. The Business Process Execution Language for Web Services (BPEL4WS or BPEL for short) was proposed by BEA, IBM and Microsoft, which represents a convergence of two languages: the Web Services Flow Language (WSFL) of IBM and XLANG of Microsoft. BPEL often acts as a description language of web service compositions. As a concurrent program language, BPEL has some special features that raise special challenges for testing[1], such as compensation handling, correlation and death-path-elimination.

Most works of concurrent program testing are based on analysis of the program reachability[2,3], that is, construction and analysis of reachability graph(RG) of the program under test. RG presents all possible states of which the program uncertainly execution reaches. But RG generation suffers from the problem of state space explosion.

Some test generation methods based on path analysis are also proposed for testing sequential and concurrent[5,6] programs. These methods firstly select local paths for individual tasks, then compose global paths with these local paths. They are applicable to programs consisting of communication processes or tasks, like those coded in Ada or CSP, but inappropriate for BPEL, which has neither explicit separation of individual processes nor synchronization via rendezvous.

Furthermore, with unique features in both syntax (e.g. flow with activity synchronization, join condition) and semantics (e.g. dead-path-elimination), BPEL needs special treatments in testing.

This paper proposes an approach to BPEL test case generation, which effectively deals with BPEL special features. To avoid state space explosion, this approach will not construct a RG and not cover all serialized paths of the program under test. Instead, this approach is based on CP-nets models, and only covers the program executable units, into which some serialized paths merged. For a program with complicated variables sharing and process interaction, this method is suitable and meets requirements of BPEL practices.

This paper is organized as follows. The modeling and analysis of a web service composition are presented in the next 2 sections, while the definition of program executable unit are given out in them. Coverage criteria for testing BPEL programs are introduced in section 4. Some algorithms which will be employed in test case generation are given in section 5. A new algorithm of BPEL test case generation is expressed in section 6. Section 7 follows with an example to illustrate our method. Section 8 concludes the paper with the compare to our formal work and the future work predictions.

II. Modeling Web Sevice Composition

To avoid ambiguity in comprehension of web service compositions coded by BPEL, we should model web service compositions, in other words, model BPEL program and analyze it. By comparison with some model ways, CP-net is employed in our works.

A. CP-nets

CP-net extends the Petri net in definition of data types and manipulation of data values. For a same system, its CP-net model should be simpler and more compact than its Petri net model in general. CP-net has been used to describe some properties, for example, security

doi:10.4304/jsw.9.3.589-595

property[7]. Petri net to analyze web service compositions [8].

B. Modeling BPEL Program with CP-nets

Based on CP-net, processes of Web service composition described by BPEL are translated into timed CP-net models in this paper, which has given in our other papers[9, 10]. Atomic activities of BPEL are seem as atomic operation in transitions, their execution successfully or not is the only factor under considered, which means the factor results in errors in not under considered, because we focus verification on web service compositions.

All CP-net models appear in this paper are generated by CPN tools[11] developed by Danish Aarhus university. The color set "STATE" has bool type. Symbol "t" represents a bool constant "true", which denotes resources movement by representing a state of an activity. "MSG" is a color set of string type, "msg" is a variable of MSG. Details see Fig.1.

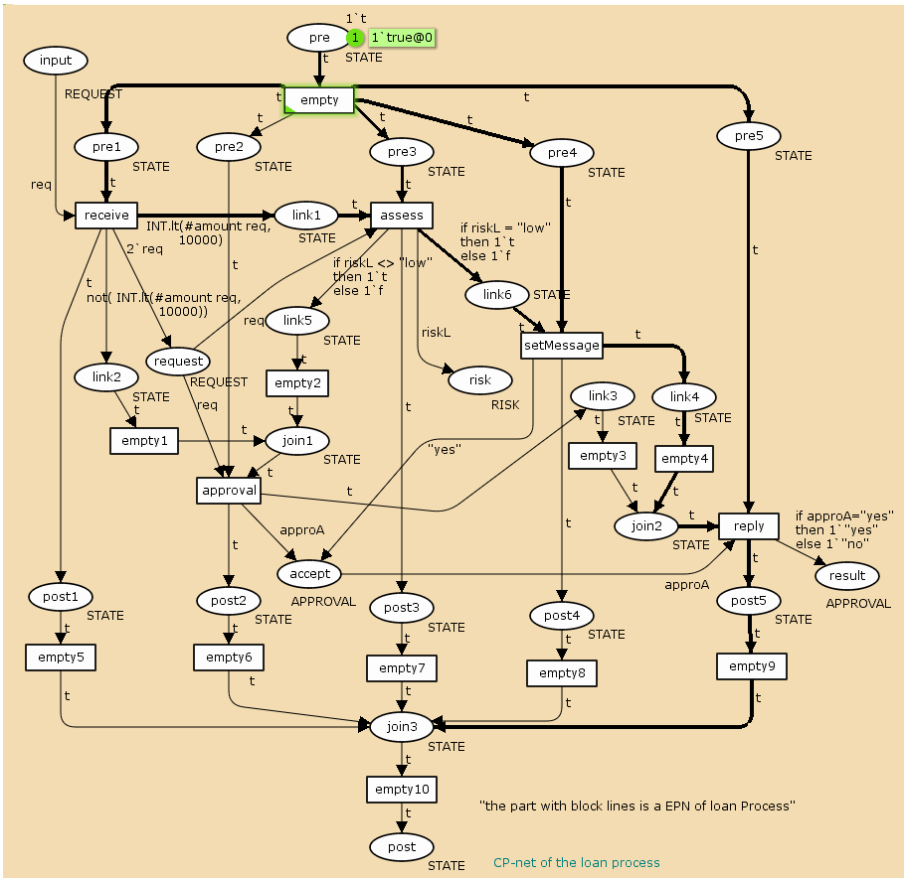

Fig.1. the CP-nets model of the Loan process with the bold parts present a PEU

III. Analysis of Model of Web Sevice Composition

With all places and transitions of CP-nets seen as vertexes, a CP-nets model can be seen as a direct graph which each arc attached an expression. This kind of graph can be treated as an acyclic one if the 0-1 criterion employed for loop structures. We do not explicit distinguish graph and CP-nets model below.

Some elements of CP-nets model M are defined below for easily description our analysis[12].

Definition 1. (Start Point and End Point) With STATE type, a place only has output arc is called a start point, the one only has input arc is called an end point of the model. Denoted by SP and EP, respectively.

Definition 2. (State point and Data point) A place with STATE type is called a state point, and the place with other types is called data point.

Definition 3. (Multiple Entering Edges Transition,MEET; Multiple Outer Edges Transition,MOET) MEET means of a transition at which concurrent paths rendezvous. Similarly, MOET means of a transition at which concurrent paths scatter. It is the vertex in the graph has more than one out edges.

Definition 4. (MEET's saturation) The in-degree of the vertex with a MEET corresponds in the graph, is called the MEET's saturation of the transition.

Definition 5. (Conditional Paths, Uncompleted Paths) The conditional paths between vertexes v_i and v_j is a sub-graph of the model M. It has sole start point v_i and sole end point v_j, and its other vertexes are all on the

paths from v_i to v_j.(the term path has same meaning as it in the theory of graph). If existing a vertex which expresses a transition of the CP-nets model and whose in-degree less than the transition's saturation in M, we say this conditional path is uncompleted, in other words, this path is an uncompleted path.

Definition 6. (Concurrent-able Paths) For paths, L_1 and L_2, if they both come from the start point and stop at the end point EP, satisfy follows:

(1) L_1 is not same as L_2.

(2) There is a transition both in L_1 and L_2, its in-degree in paths less than its saturation in the model M. we denoted it by tran.

(3) L_1 and L_2 have a common conditional path from tran to the EP.

We call L_1 and L_2 are concurrent-able paths.

Obviously, two concurrent-able paths should rendezvous at same vertex, tran, then their remainders merge into a same portion. Here the algorithm of merging them is given as Algorithm 1.

Algorithm 1($Concurrentable(L_1, L_2)$,Check two paths are concurrent-able or not)

Input: paths L_1 and L_2

Output: if L_1 and L_2 are concurrency, returns true, else false.

Steps:

(1) With same vertex set and vertex order as those of model M, present L_1 and L_2 by their adjacent matrices A_1 and A_2, respectively.

(2) Take AND operation on elements which on same place in matrix A_1 and A_2, results in a matrix A.

(3) Find the serial number of vertex with which the EP correspond, denotes it by p.

(4). check the p th column of A.

if (existing one and only one t satisfies: at the location of the t th row the p th column, the element of A is not zero) {return *true* . }

else {return *false* }.

In the step (4), the condition "existing one and only one t satisfies: at the location of the t th row the p th column, the element of A is not zero" means the items (2)and (3) of definition met.

Algorithm 2($Merge(L_1, L_2)$,Merging two concurrent-able paths)

Input: concurrent-able paths L_1 and L_2

Output: path L into which L_1 and L_2 merged.

Steps:

(1) With same vertex set and vertex order as those of model M, present L_1 and L_2 by their adjacent matrices A_1 and A_2, respectively.

(2) Take OR operation on elements which on same place in matrix A_1 and A_2, results in a matrix A.

(3) The path expressed by the matrix A is the merged path L.

Definition 7. (Program Executable Units, PEU) for a model M, a program executable unit is a sub-model that satisfy follows:

(1) The SP and EP of M are belong to this sub-model.

(2) Any transition is saturation,, or any vertex corresponding to a transition has same in-degree than it in model.

Arcs in a graph are divided into 2 categories: process control arc and data transmit arc. In follows, PEU we refer to is the one that generate from the model which all data flow ignored. In Fig.1, the bold line parts present a PEU of the model of Loan process[1].

IV. Testing Coverage Criterion

A good test coverage criterion should meet two aspects. One is the amount of required synchronous sequences which reaches the criterion in a reasonable range, and another is the synchronous sequences selected by this criterion is benefit to find errors of the concurrent program under test[13-15].

A. Existing Coverage Criteria for BPEL Program Testing

To measure the quality of software testing, some coverage criteria are given and some metric tools are developed. But for the unit test quality of the business process written in BPEL, there is no uniform measure criterion up to now.

Existing coverage criteria for BPEL process are divided into the following five categories: activity coverage, branch coverage, link coverage, fault handler coverage, and compensation coverage[16-18]. L.Daniel extended the test tools BPELUnit to support these five measures[19]. D.Lubke also gave out an instrumental method to obtain values of these five measures[18].

B. A New Coverage Criteria for BPEL Program Testing

As previous definition, a program executable unit implements in one execution. The amount of units covered is what we consider in this paper while test case generating for a web service composition. For loop structure, 0-1 criterion employed. While test case generating, we strive to cover all such units. That is PEU coverage criterion we adopt for the BPEL program testing.

For a CP-nets model of a web service composition, this criterion includes place coverage, transition coverage; and for a BPEL program, it includes activity coverage, concurrent executable path coverage, and also link coverage.

V. Some Algorithms related to Test Case Generation

To generate test case of BPEL programs is complicated Obstacles need to over come. In this section, some algorithms are designed to solve problems during test case generation.

A. An Algorithm of PEU Generation

To generate test cases satisfy the test criterion of BPEL concurrent paths, we need to obtain all PEUs. For two PEU, it is possible to execute asynchronous concurrently in a running.

Algorithm 3 (Generation PEU base on control flow)

Input: the CP-nets model of a BPEL program

Output: E, a set of all PEUs of the program

Steps:

(1) Simplify the CP-nets model of the BPEL program as one which only has control flows, that is to remove out vertexes and edges presenting data flow. Denoted by M.

(2) Initialize the set, E , of all PEUs to empty.

(3) Traverse M to generate all conditional paths which from SP to EP. Use *CSPaths* to denote the set of these paths.

(4) If *CSPaths* is not empty, for each path $L \in CSPaths$, if L is a PEU, then let $E = E \cup \{L\}$, $CSPaths = CSPaths \setminus \{L\}$.

(Note: After this step, if *CSPaths* is not empty, then any path of it has uncompleted concurrent-able path in *CSPaths* .)

(5) while (*CSPaths* is not empty)

{ for each pair of paths L_1 and L_2

{ if ($Concurrentable(L_1, L_2) = true$)

{ let $L = Union(L_1, L_2)$;

if L is a PEU, let $E = E \cup \{L\}$;

else let

$CSPaths = CSPaths \cup \{L\}$,

$CSPaths = CSPaths \setminus \{L_1, L_2\}$ }

}

}

(6) Output E .

B. Algorithms of Dealing with Constraints

The PEU we mentioned above is the essential unit which will be dealt with in our approach.

After found the unit required to test, the things should to do is attach data on them, automatically or manually. Some methods are available, for example, all constraint conditions are sent into a constraint system, which result in test cases. The test data we obtained is input messages that may include output which needs revised manually. When no output included, it is need to supply manually. When some information is absent from the constraint conditions, it can be treated as a free variable and its values can be generated randomly.

Based on the CP-nets features, a new algorithm of dealing with constraints is presented as follows. It is combined with the reverse replacement method which proposed by J. Zhang[20] and improved by J. Yan[21].

Algorithm 4 ($predicatesConstra\,\mathrm{int}(Node\,N, PEU\,epu)$,

Calculate the constraint set of node N of PEU epu)

Input: a PEU *epu* and a node N of the model

Output: the condition which N meets while *epu* running, $constra\,\mathrm{int}(N)$

Steps:

(1) if N is a place, use p to denote N .

if(${}^{\bullet}p == \phi$)

{ if($(I(p) \geq \sum_{t \in p^{\bullet}} E(p,t))$=*false*)

$\mathrm{constraint}(\mathrm{p}) = \{false\}$

elseif ($(I(p) \geq \sum_{t \in p^{\bullet}} E(p,t))$=true)

$\mathrm{constraint}(\mathrm{p}) = \{\}$

else $\mathrm{constraint}(\mathrm{p}) = \{I(p) \geq \sum_{t \in p^{\bullet}} E(p,t)\}$

}

else

{ if($(\sum_{t \in {}^{\bullet}p} E(t,p) \geq \sum_{t \in p^{\bullet}} E(p,t)) = false$)

$\mathrm{constraint}(\mathrm{p}) = \{false\}$

elseif ($(\sum_{t \in {}^{\bullet}p} E(t,p) \geq \sum_{t \in p^{\bullet}} E(p,t)) = true$)

$\mathrm{constraint}(\mathrm{p}) = \{\}$

else

$\mathrm{constraint}(\mathrm{p}) = \{\sum_{t \in {}^{\bullet}p} E(t,p) \geq \sum_{t \in p^{\bullet}} E(p,t)\}$

}

Here ${}^{\bullet}p$ and $p^{\bullet}$ is the preset and postset of p , respectively.

(2) if N is a transition, use t to denote N .

if($G(t) = true$)

$constra\,\mathrm{int}(t) = \{\mathrm{var}1, \mathrm{var}2, \cdots, \mathrm{var}l\}$

else $constra\,\mathrm{int}(t) = \{G(t), \mathrm{var}1, \mathrm{var}2, \cdots, \mathrm{var}l\}$.

Here G(t) is a guard function of transition t , $\mathrm{var}1, \mathrm{var}2, \cdots, \mathrm{var}l$ is a variable sequence the transition t output when t fired.

(3) Output $constra\,\mathrm{int}(N)$.

The result of this algorithm, $constra\,\mathrm{int}(N)$, is a sequence whose elements are predicates or variables. Employing this algorithm, the constrains set for each node of the PEU can be obtained.

In the follow algorithm 5, we adapt the breadth-first research with a slight adjustment in order, according to the in-degree of each vertex. The constraint condition of each vertex is appended to the set constraintMerge by its order. For a transition vertex which has multiple enter edges, the adoption of breadth-first research and the way which vertex will not be dealt with until its in-degree is 1, ensured the condition attached on in-edge of the vertex is appended to constraintMerge earlier than the transition guard function. This correspond to the semantic of firing of transition in CP-nets. The constraint set generated, constraintMerge, is a sequence.

Algorithm 5 (

*constra*int *Collect*(*PEU epu with* Constrain of each node) , Calculate the constraint set of epu)

Input: a program executable unit epu , and the constraint(N) of each node N in epu .

Output: the constraint condition set of epu needs when it running.

Steps:

(1) Calculate the in-degree of each node N , denote it as in(N) .

(2) Denote the start point of *epu* as *v* .

```
constraintMerge = PredicatesConstraint(v， epu) ,
visited[v]=true;
IniQueue(Q);   // initialize Q as empty
EnQueue(Q,v);   // v enter the queue Q
```

(3) while(Not Empty(Q))

```
{  // the element Q move out the queue, denoted by v
   v=DelQueue(Q)
   w= firstAdj(v)  //calculate the adjacent vertex of v
   while(w!= 0)  //if the adjacent vertice of v
   {   if(not visited[w])
       if(in(w)= =1)
       {    constraintMerge= onstraintMerge Appened constraint(w);
         // appended elements of constraint(w) to the
         //constraintMerge sequentially
            visited[w]=true;
            EnQueue(Q,w);  //w enter the queue Q
       }   //endif
       in(w)=in(w)-1;
       //get the next adjacent vertex of v
       w=nextAdj(v,w);
   }   //endwhile
}  //endwhile
```

(4) Output the set constraintMerge.

For each valid EPU, we will get a constraint condition set. For different EPUs, their sets maybe are identical, or has an inclusion relation. For example, "loanAmount <10000" and "loanAmount <10000 && riskL="low" ". so filtrition is needed to reduce test cases in this situation.

Algorithm 6 (Filtrition of Constraint Sets)

Input: *CES* ,the sets of constraint sets for all valid EPUs.

Output: *NCES* , the sets of constraint sets filtered.

Steps:

(1) let *size* denotes the size of *CES* .

(2) for each constraint set, *ces*1

```
    for each constraint set, ces2
        if ( ces1 and ces2 are identical )
            remove ces2 out CES .
```

(3) for each constraint set, *ces*1

```
    for each constraint set, ces2
        if ( ces1 includes ces2 )
            remove ces1 out CES .
        elseif ( ces2 includes ces1 )
            remove ces2 out CES .
```

(4) Output the set *CES* .

After filtrition, each pair of constraint sets have no intersection parts..

C. Remove out Invalid PEU

For the PEUs obtained based on control flows, it is not ensure to be able to run. The semantic of an enable transition shows: if an un-enabled transition exists in a PEU, this PEU can not execute. We can ignore this kind of PEU during the test data generated, that is these PEUs should be removed out. Some PEUs also can not execute if no initial values to ensure the program run along it.

Algorithm 7 (Remove out invalid PEUs)

Input: E ,the set of PEU. constra int(N) ,Constraint set for each node

Output: E with some invalid PEU removed

Steps:

(1) for each $epu \in E$

if (exist a node N in epu, satisfies $false \in constra\,\mathrm{int}(N)$)

$E = E \setminus epu$ //remove out *epu* from E

(2) Output the set E .

This algorithm deals with CP-nets models locally, so only those PEUs, which are invalid to test, can be removed out. For a PEU without resolution of the constraint set, it can not execute and needs remove out.

VI. A ALGORITHM OF BPEL TEST CASE GENERATION

In this Section, we will elaborate the proposed BPEL test case generation method. This method contains seven steps as follows:

Algorithm 8 (BPEL test case generation).

Input: a BPEL program

Output: test cases

Steps:

(1) Model the BPEL program to CP-nets M .

(2) Generate the set of all PEUs of the model M , denoted by *PS*

(3) Remove the invalid PEUs from *PS* .

(4) Generate the constraint set for each EPU of *PS* .

(5) Calculate the constraint set for each EPU of *PS* , the solution denoted by *s* . All solutions are denoted by *CES* .

(6) Remove *s* which is empty out *CES* , that is remove invalid EPUs further.

(7) Filter the Constraint Sets

(8) Derived test data from each $s \in CES$. Denoted by *d*

(9) Derived test cases from *d* .

Test cases can be in some forms. XML is employed in our work, in which a test case is described as a TestUnit element with an attribute of TestData. This kind of formal description can be input a test tool automatically.

VII. TEST CASE GENERATION OF WEB SERVICE COMPOSITION

The example of Loan Approval Process[1] is used to illustrate our approach. This example consists of a simple loan approval service. Customers of the service send loan

requests, including personal information and amount being requested. Using this information, the loan service executes a simple process resulting in either a "loan approved" message or a "loan rejected" message. The decision is based on the amount requested and the risk associated with the customer. For low amounts of less than $10,000 a streamlined process is used. In the streamlined process low-risk customers are approved automatically. For higher amounts, or medium and high-risk customers, the credit request requires further processing. For each request, the loan service uses the functionality provided by two other services. In the streamlined process, used for low amount loans, a risk assessment service is used to obtain a quick evaluation of the risk associated with the customer.

The CP-nets model of this web service composition is shown is Fig.1.

15 sequence test paths are produced by our algorithm, and 7 program executable units derived from them. The constraint sets of these 7 units are following.

loanAmount >=10000;
loanAmount >=10000;
loanAmount <10000;
loanAmount <10000 && riskL="low";
loanAmount <10000 && riskL="low";
loanAmount <10000 && riskL!= "low";
loanAmount <10000 && riskL!= "low".

After the filtrition, only 3 of them retained.

loanAmount >=10000;
loanAmount <10000 && riskL="low";
loanAmount <10000 && riskL!= "low";

With respect to equivalence class, above 3 cases include all classes. The number of PEUs need to test is less than 9 which is the minimum available in reference[16]. Further more, the test cases we need is far less than formal works.

VIII. Conclusion

A new approach of test case generation of web service composition is expressed. It can deal with BPEL features properly. Such as concurrent, DPE,etc. This approach is based on model techniques, and can be combined with other testing techniques to use. For example, after improved slightly, it can be used to test some programs written in other business process languages, such as BOMN, XPDL, XLANG, ESFL, etc. with other test criterions adopted. These works are improvement of our previous works[12]. It has advantages as follows.

(1) The analysis of CP-nets models directly avoids state space explosion.

(2) It well adapts to programs of little variables sharing, or concurrent, especially. Because not too much interaction between the concurrent actives.

(3) The application of filter on constraint sets and the improvement of algorithms reduce the test cases number.

(4) The PEU we defined can run directly in the program running environment, and can be used to explore more program errors relate to concurrent.

Compare to our previous work, we design some algorithms to check paths are concurrency or not, and filter constraint sets of PEU. We also improve the generation of test cases, which results in less test cases.

The further research work should be done in two aspects. The variables of the model needs further analysis, the data dependence and control dependence should be considered together, an improved generation method of test case for programs with multiple variable sharing will proposed. Another aspect is that some high level features of BPEL, such as scope nest, needs to be dealt with.

Acknowledgements

This work was financially supported by the Department of Education of Shaanxi Foundation (2013JK1190).

References

[1] OASIS. Web Services Business Process Execution Language Version 2.0. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf. 2008.
[2] L. Chen. “Automatic Test Cases Generation for Statechart Specifications from Semantics to Algorithm”. Journal of Computers. 2011, Vol. 6(4):769-775
[3] Y. Yuan, Z.J. Li, W. “Sun. A Graph-Search Based Approach to BPEL4WS Test Generation”. International Conference on Software Engineering Advances (ICSEA'06),2006:14-20
[4] R.N. Taylor, D.L. Levine, and C.D. Kelly. “Structural testing of concurrent programs”. IEEE Transactions on Software Engineering. March, 1992, vol18(3):206–215.
[5] R.D. Yang and C.G. Chung, “A path analysis approach to concurrent program testing”, Information and Software Technology, 1992, 34(1): 43-56.
[6] T. Katayama, E. Itoh, and Z. Furukawa, “Test-case generation for concurrent programs with the testing criteria using interaction sequences”, Proceedings of the 6th Asian-Pacific Software Engineering Conference, December 1999:590-597.
[7] Y. Xu, X. Xie. Modeling and Analysis of Security Protocols Using Colored Petri Nets[J]. Journal of Computers. 2011, Vol 6(1):19-27
[8] G. Fan, H. Yu, L. Chen, and T. Ruan. “An Approach to Analyzing Time Constrained Service Composition”. Journal of Computers. 2011, Vol 6(8):1723-1731
[9] Y. WANG, P. CHEN. “Models of BPEL"s flow activity based on color Petri net”. Application Research of Computers,2011,28(2):631-634.
[10] Y. WANG, P. CHEN. “Models of Web Services Composition Based on Timed Color Petri Nets” [J]. COMPUTER SCIENCE,2010,37(10):pages151-155
[11] http://wiki.daimi.au.dk/cpntools/cpntools.wiki[OL]. 2009
[12] Y. WANG, P. CHEN. “Test Case Generation of Web Service Composition: an Approach Based on the Color Petri Net”. Applied Mechanics and Materials .2013,Vol. 336-338: 2063-2070
[13] J. Thomas. McCabe. “A Complexity Measure”. IEEE Transaction on Software Engineering. 1976, SE-2(4):308–320.
[14] R. D. Yang and C. G. Chung. “A path analysis approach to concurrent program testing. Information and Software Technology”. 1992, Vol34(1): 43–56
[15] L. Mei, W. K. Chan, and T. H. Tse. “Data Flow Testing of Service-Oriented Workflow Applications”. Proceedings of

the 30th International Conference on Software Engineering (ICSE 2008), New York, NY, USA: ACM, 2008:371–380

[16] C. Ouyang, E. Verbeek, and W. M. P. Van der Aalst et al. “Formal semantics and analysis of control flow in WS-BPEL”. Science of Computer Programming archive. July 2007, Vol67(2-3):162-198

[17] D. Lubke and A. Salnikow. “Definition and Formalization of BPEL Process Test Coverage”. Technical report, Leibniz Universit¨at Hannover, FG Software Engineering. http://www.se.uni-hannover.de/techreports/2009-01Definition AndFormalizationOfBpelProcessTestCoverage.pdf, 2008.

[18] D. Lubke, L. Singer, A. Salnikow. “Calculating BPEL Test Coverage through Instrumentation”. Proceedings of the 4th International Workshop on Automation of Software Test, AST 2009, Vancouver, BC, Canada: May 2009.

[19] M. Phili. BPELUnit. URL http://www.bpelunit.org. 09/01/2007.

[20] J. Zhang, X. Wang. “A constraint solver and its application to path feasibility analysis”. International Journal of Software Engineering & Knowledge Engineering. 2001. Vol.11(2):139–156.

[21] J. Yan, Z. Li, Y. Yuan et al. “BPEL4WS Unit Testing: Test Case Generation Using a Concurrent Path Analysis Approach”. In 17th International Symposium on Software Reliability Engineering (ISSRE’06). IEEE, 2006

Yuying Wang was born in Chifeng, Inner Mongolia, China in 1964. She awarded a BS degree in mathematics from Beijing Normal University in 1987, China. She received the MS and Ph.D degrees in software engineering from Xidian University, Xi'an, China in 2003 and 2012. She is an associate professor of math and software engineering, Xian University of Architecture and Technology, China. Her research interests include model checking, data mine and optimization.