A Validation Framework for the Service-Oriented Process Designing

Guoqiang Li[1,2], Lejian Liao[1], Fuzhen Sun[1]
[1]Beijing Engineering Research Centre of High Volume Language Information Processing & Cloud Computing Applications,
Beijing Key Laboratory of Intelligent Information Technology,
School of Computer Science, Beijing Institute of Technology, Beijing, China
[2]School of information, Linyi University, Linyi, China
{lgqsj, liaolj, 1090723}@bit.edu.cn

***Abstract*—In the service-oriented software systems, the services composition process is modeled using the service orchestration languages whose fault-handling and compensation mechanisms are crucial to guarantee the process running successfully. In this paper we propose to extend the syntax of BPEL to improve these two mechanisms. In order to validate their correctness, the composition is transformed to the planning graph. Then the validation of the fault-handling mechanism is regarded as a problem of seeking solution from the solution sets gained from the planning graph. We analyze the services composition structures and construct a relationship matrix to complete the validation of the compensation mechanism. A validation framework is proposed and an experiment is implemented to show our method effectiveness.**

Keywords - graph planning; BPEL; service orchestration; fault handling; compensation mechanism

I. INTRODUCTION

Service-oriented paradigm is capturing a growing interest as a mean for business to business integration. To realize the composition of the web services, researchers and industrial practitioners have proposed several web service orchestration languages such as BPEL4WS [1], WSFL [2], XLANG [3] and StAC [4]. And BPEL4WS is the de facto standard. Compare to the other languages, BPEL4WS supports this problem with a programmable and scope-based fault-handling and compensation mechanisms. Fault-handling mechanism guarantees the composition continues to achieve the goal. The function of compensation mechanism is to maintain the consistency of the whole process by eliminates the effects of everything executed from the failed service. But, it is a time-consuming and error-prone task to design these strategies and it is difficult to validate the correctness by the designers completely.

In order to solve the above problem, we propose to make use of graph planning technology focusing on the correctness validation of the business process during the design phase. The contribution of this paper includes:

- Syntax of the BPEL is proposed to extend with two operators corresponding to the fault-handling and compensation-handling mechanisms and their semantics are presented. A planning graph is constructed by means of analyzing the business process.
- For the fault-handling mechanism, it is transformed to seek a solution from the solution set of graph.
- Analyze the structural relationship of services and build a relationship matrix to facilitate the validation of the compensation-handling mechanism.

This paper is organized as follows. In the next section, we place the related work. Section Ⅲ introduces the extension of BPEL and the graph planning technology. The Section Ⅳ details the validation framework including the validate algorithms. The experiment is implemented and analyzed in Section Ⅴ. The conclusion of this paper and future work are discussed in the Section Ⅵ.

II. RELATED WORK

To guarantee the correctness of the business process, many researchers consider the semantic model. A simplified version of the WS-BPEL is defined in [5]. Compensation closure and context are proposed to capture the execution structure and form a good framework to the semantics of implementation of BPEL4WS. In [6], Chenguang, Shengchao and Zongyan verify the process using the Hoare-logic. In [7], Huibiao, Jifeng, Jing and Bowen focus on deriving the operational semantics and denotational semantics from algebraic semantics. Algebraic laws for BPEL programs are considered. Comparing with these methods, our approach is more intuitive.

A logic model specifies the semantics of workflows and composite tasks are given in [8]. A set of inference rules are presented to deduce the strongest post condition and weakest precondition and automatic workflow verification is demonstrated. The interactions of composite web services are modeled as conversations in [9]. The guarded automaton augmented with unbounded queues for incoming messages is used to be the intermediate representation and the model checker SPIN verifies synchronous communication. But, it is a challenge to translate the BPEL to the Promela program which is the input of SPIN for the designers. A Petri-net based formalization to construct composition process is proposed in [10]. And the interface dependency, compensation dependency and sequence triggered in nesting scopes are discussed. These preceding methods focus on the validation of the fault-handling and compensation-handling during the running phrase. From a transactional perspective of the compositions, many works introduce their approaches [11-13], e.g., a heuristic-based analysis of the process definition is proposed in [11]. The analysis result is a set of nonrepairable activities, whose impacts are evaluated by a repairability reasoner. Then a combination of the fault and the branching probabilities associated with an activity is given to gain a relevance index, which is used to remind the designer of knowing that to improve the repairability of the process. A transactional service patterns are used in [12] to specify the transactional composite service (CS) using Event Calculus. The CS transactional behavior is specified initially by the designers. Then the patterns and transactional flow are rewritten using EC predicates. Last, the behavior consistency is checked according to the predefined transactional consistency rules. Similar to our work, a planning graph is also used in [14], which only considers the repair technique for the composition adaptation rather than validating the correctness of the composition. A testing tool for web services composition is proposed in [15]. This tool focuses on conformance testing and unit testing considering the timing constraints and synchronous time delay. But, the activities of a flow activity are processed as sequence activities instead of processing in parallel in this tool. A web services translation tool is proposed in [16]. This tool is used to design and verify a web services system with time restrictions during the design phase. UML is used in the design phase to model the system to provide sequence diagram, which is transformed to choreography description by WS-CDL. Last, the UPPAAL tool is used for validation and verification purpose. An on-line approach is introduced in [17] to test an orchestration of web service composition and a passive testing verifies a timed trace with respect to a set of constraints. But, it does not pay close attention to the fault-handling and compensation-handling mechanisms.

III. EXTENSION OF BPEL AND GRAPH PLANNING

A. Syntax and Semantic of the Extended BPEL(ex-BPEL for short)

ex-BPEL builds on the base of the BPEL by extending the original fault handling mechanism. A business process (BP) includes four components: an activity P, a basic activity A, a fault handler F and a compensation handler C. The detailed syntax is as follows:

```
    BP:= 【P: F】
    P:= A        (basic activities)
     | skip            (do nothing)
     |P; P             ( sequence)
     |P || P           (flow)
     | if b then P else P   (conditional)
     |n: {P?C:F}        (scope)
    A:= e            (assignment)
     | rec p y         (receive)
     | inv p x y         (invoke)
     | rep p x          (reply)
     | throw           ( throw a fault)
 C, F := ↤n   ( compensation)  |  retry P: N | substitute P: P’
```

The operational semantics of P and A are same as the semantics of [6]. For example, sequence “P_1; P_2” presents an order of these two activities, i.e., P_2 starts running only after P_1 completes.

The extension of the fault mechanism includes two new operators: *retry* and *substitute*. The operator *retry* $P:N$ means activity P makes N repetitions and *substitute* $P_1:P_2$ means P_1 substitutes P_2 if P_2 fails. Actually, the two operators can be combined to describe complex handling strategy.

A work-through scenario is an e-travel example. To plan to travel from place A to B, a train ticket should be ordered first and another choice is to book a flight ticket if no train ticket. Then a hotel should be booked. In case of hotel booking failure, we can re-order the ticket or cancel the plan.

More details of the fault handling and compensation handling syntax of BPEL is referred to [1].

B. Extension of Fault-handling Mechanism

We distinguish two types of faults: temporary faults and permanent faults. For example, a temporary fault may be a network interruption in a short time. After the fault is thrown from the business process, firstly we analyze the type of the fault, and then we choose the handling mechanism for it. *Retry* is used to cope with the temporary faults, and *substitute* handles the permanent faults. So, the modified fault handler is as follows:

```
<faultHandlers>
<catch faultName="FailofTrainTicket"?
faultVariable="ncname"? >
<retry><invoke partnerLink=”TrainSupplier”
portType=”Trainsup:OrderInterface”
Operation=”submitOrder” inputVariable = “OrderInfo”
outputVariable= “OrderConfirmation”>[N]
</retry>
<substitute> <invoke partnerLink=”FlySupplier”
portType=“Flysup:OrderInterface”
Operation=”submitOrder”  inputVariable = “OrderInfo”
outputVariable= “OrderConfirmation”>
</invoke> </substitute></catch> </faultHandlers>
```

“[N]” specifies the number of repetitions in *retry* operation.

C. Introduction to Graph Planning

A planning graph is a directed, leveled graph with nodes and edges, denoting as $\langle V, E \rangle$ [18]. $V = \langle Prop, Action \rangle$, $Prop$ is a set of all proposition levels $\{\mathrm{Prop}_0, \mathrm{Prop}_1, \mathrm{Prop}_2 ... \mathrm{Prop}_n\}$, *Action* is a set of all action levels $\{\mathrm{Action}_0, \mathrm{Action}_1, \mathrm{Action}_2 ... \mathrm{Action}_n\}$ where an action is described as: $Action = (name(Params, Pre, Add, Del))$, where Pre specifies the preconditions of this action and Add specifies its positive effects. While the Del specifies its negative effects. The proposition levels and the action levels occur alternately. So, the planning graph is: $\{\mathrm{Prop}_0, \mathrm{Action}_0, \mathrm{Prop}_1, \mathrm{Action}_1 ... \mathrm{Prop}_n\}$ shown in Fig. 1, where Prop_0 specifies the initial proposition level and Prop_n specifies the goals proposition level. If one proposition Prop0i exists in Pre of one action A, then there is an edge between Prop0i and A. Similarly, if one proposition Prop0j exists in Add of one action B, then there is an edge between Prop0j and B.

In Fig. 1, the black circle is a proposition node and the rectangle is an action node. The dotted line means every proposition that appears in proposition-level i may also appear in proposition level i+1, allowed by “no-op actions”. Because of this trait, action-level i may contain all the possible actions whose preconditions all exist in proposition-level i [18]. So, for the *retry* operation, we can determine the maximum occurs times according to the N of a service when it fails. As shown in Fig. 1, the grey rectangle means the services ws_1 and ws_i should be updated in that level. We will not distinguish the action from service from here.

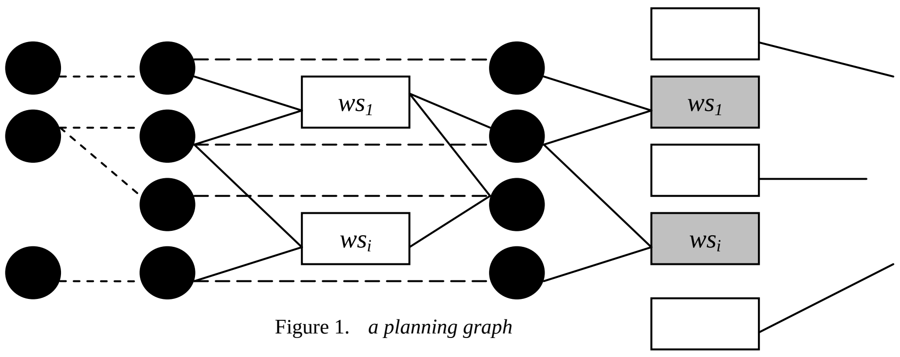


Figure 1. *a planning graph*

IV. VALIDATION FRAMEWORK

There are three modules in our validation framework as shown in Fig. 2. The Parsing module includes BPEL Parser and WSDL Parser. The former parses the BPEL documents and gets the service structure relationship matrix which is stored in the database. The latter parses the WSDL documents to get the corresponding actions. The Graph Planner is used to gain the solutions.

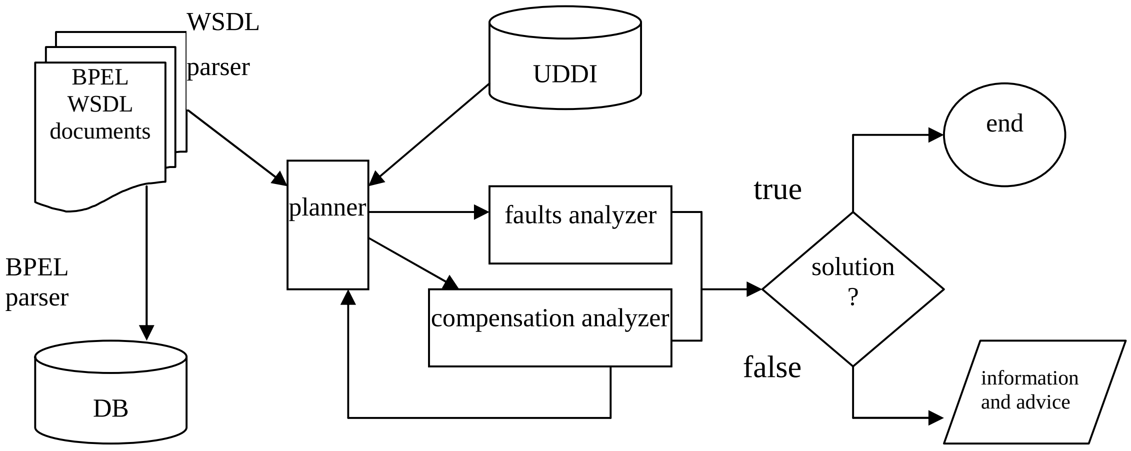


Figure 2. the validation framework

A. *Parsing Modules*

The parsing modules are responsible for generating the original input data.

1) *WSDL Parser*

WSDL is an XML format for describing web services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. WSDL Parser transforms the services description to the actions presented by STRIPS, the algorithm is as follows:

input: WSDL documents: $\{Wsdl_1, Wsdl_2\ldots\ldots\}$
output: a set of actions of planning: $\{Action_1, Action_2\ldots\ldots\}$
procedure:
(a) name = the names appear in label: <wsdl:service></wsdl:service> of $Wsdl_i$
(b) Params = the list of all the names of the labels: <wsdl:message></wsdl:message>of $Wsdl_i$.
(c) Pre = the conjunction of all <wsdl:input></wsdl:input> as defined in label<wsdl:binding></wsdl:binding> of $Wsdl_i$.
(d)Add = the conjunction of all <wsdl:output></wsdl:output> as defined in label <wsdl:binding></wsdl:binding> of $Wsdl_{i.}$
(e) return $Action_i$ = (name(Params, Pre, Add)).

2) *BPEL Parser*

There are two ways in which two actions are marked to be exclusive of each other: (1) interference: if either of the actions deletes a precondition or add-effect of the other; (2) competing needs: if there is a precondition of action A and a precondition of action B that are marked as mutually exclusive of each other in the previous proposition level [18].

We treat the compensation information as the *del* information corresponding to the attribute "*del*" of the original *STRIPS* representation. Suppose *ws* is a service in business process, the set of services in its fault handler is defined $Wsf = \{wsf_1, wsf_2 ...\}$ and the set of services in its compensation handler is $Wsc = \{wsc_1, wsc_2 ....\}$. So, its mutual exclusion set is $Mes = Wsc$. All this information is stored in database and used in graph planning.

B. Implements of Validation

1) Construction of Planning Graph

It is simple to transform the business process to a planning graph. The actions of every level correspond to the services of one step of the process.

2) Definitions of Validation Properties

The validations of the fault and compensation handling mechanisms require all the satisfied solution i.e., the actions sequence: $S = \{S_1, S_2, S_3 ...\}$. We give some definitions on the validation properties.

a) fault-amendable service: for every service *ws*, if ws' exists and satisfies: $ws' \in Wsf$ and $ws' \in S$, we say this service fault-amendable.

b) fault-amendable process: if all the services of a process are fault-amendable, we say the process is fault-amendable.

c) compensation-amendable service: for every service *ws*, if ws' exists and $ws' \in Wsc$ can bring the process to a consistent state, we say this service compensation-amendable.

d) compensation-amendable process: if all the services of a process are compensation-amendable, we say the process is compensation-amendable.

e) reliable process: if the process is fault-amendable and compensation-amendable, we say the process is reliable.

3) Validation Algorithm of Fault-handling

```
begin
for each $ws_i \in BPEL$ Process
  if $\exists$ ws $\in Wsf \wedge ws \in S$
      return true
   else return $ws_i$
end
```

If the definition (a) is satisfied, the result is true, else a fault service is returned, and a handling suggestion will be given to the designers.

4) Validation of the Compensation-handling

According to the structures of the BPEL, we define the structure relationship as follows:

a) Sequence structure: in Fig. 3, ws_1 is a directly prior of ws_2, denoted: $ws_1 \prec ws_2$. And ws_2 is a directly successor of ws_1, denoted: $ws_2 \succ ws_1$. If a compensation of ws_2 is invoked, the compensation of ws_1 should be invoked.

b) Xor structure: in Fig. 4, ws_1 is a directly xor-split prior of ws_2 denoted: $ws_1 \prec_{xs} ws_2$. And ws_2 is a directly *xor-split* successor of ws_1, denoted: $ws_2 \succ_{xs} ws_1$. ws_2 is a directly *xor-join* prior of ws_4, denoted : $ws_2 \prec_{xj} ws_4$, ws_4 is a directly *xor-join* successor of ws_2, denoted: $ws_4 \succ_{xj} ws_2$.

c) And structure: in Fig. 5, ws_1 is a directly *and-split* prior of ws_2 denoted: $ws_1 \prec_{as} ws_2$ .And ws_2 is a directly *xor-split* successor of ws_1, denoted: $ws_2 \succ_{as} ws_1$. ws_2 is a directly *and-join* prior of ws_4, denoted: $ws_2 \prec_{aj} ws_4$, ws_4 is a directly *and-join* successor of ws_2 ,denoted $ws_4 \succ_{aj} ws_2$.

d) Parallel-or structure: in Fig. 4, ws_2 and ws_3 are parallel in *xor* structure, denoted $ws_2 \|_{or} ws_3$. The pair of services will not affect each other while any of them throws a fault. In this case, if the compensation of ws_2 is invoked and ws$_3$ runs normally, the compensation of *ws$_1$* can not be invoked.

e) Parallel-and structure: in Fig. 5, ws_2 and ws_3 are parallel in *and* structure, denoted: $ws_2 \|_{and} ws_3$. If ws_2 is compensated, ws_3 must be compensated, and the compensation of *ws$_1$* will be invoked.

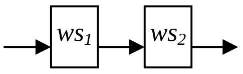


Figure 3. *Sequence structure*

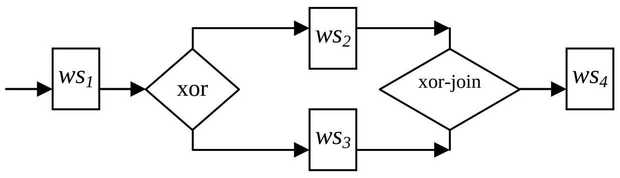


Figure 4. *Xor structure*

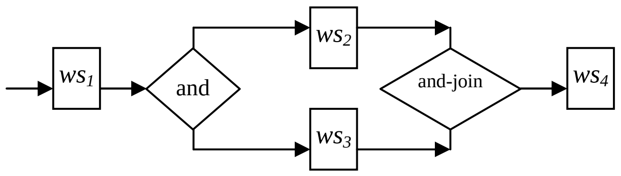


Figure 5. "*and*" *structure*

$$\begin{array}{ccccc}
 & ws_1 & ws_2 & ws_3 & ws_4 \\
ws_1 & - & \prec_{xs} & \prec_{xs} & - \\
ws_2 & \succ_{xs} & - & \|_{or} & \prec_{xj} \\
ws_3 & \succ_{xs} & \|_{or} & - & \prec_{xj} \\
ws_4 & - & \succ_{xj} & \succ_{xj} & -
\end{array}$$

Figure 6. *Relationship matrix*

All the structure relationship of the services will be stored in a matrix, which can be automatically generated in the parsing of BPEL documents. The corresponding matrix of Fig. 4 is the Fig. 6, where the symbol "-" means no relationship between services.

Suppose the compensated service is *ws* and the service relationship matrix is *ws*-matrix. The algorithm to validate the compensation-handling mechanism is as follows:

a) Step 1: Look for a set of the services which is related to *ws* until a *xor-split* service *ws-xs* or the first service is found, and the path is recorded, denoted *ws-path*.

b) Step 2: Locate the directly successor *ws-post* of *ws*.

c) Step 3: Make sure whether there is a path from *ws-post* to *ws*-xs and the path exists in solution sets, if so, the validation of compensation-handling ends.

d) Step 4: if not, take the *del* information of the compensation services on the *ws-path* as the goal propositions and do the planning. If the solution can be found, the process is reliable, else the faulty service is located and advice is given.

For example, if ws_2 is compensated, its directly successor is ws_4. Because there are two path from ws_4 to ws_1, i.e., $ws_4 \rightarrow ws_2 \rightarrow ws_1$ and $ws_4 \rightarrow ws_3 \rightarrow ws_1$. So, if the compensation handling of ws_2 is defective nevertheless ws_3 is available, the process can run successfully.

5) *Analysis of the Algorithms:*

a) For the faults-handling, the complexity is O(n), n is the number of the services which is semantic or functionally equivalent to the faulty service in the same level.

b) For the compensation-handling, the complexity is O(n^2), which is the time needed to look for a path between given two nodes in a graph. If the path does not exist, we should do a new planning process, which is at least PSPACE-hard. In spite of this, the planning graph analysis can provide a quite substantial improvement in running time [18].

V. EXPERIMENT

The goals of the experiments are: (1) To validate the soundness and completeness. Soundness is that if there is a problem in the fault-handling and compensation-handling, the system is able to find it. Completeness is that if the fault information is returned, it is related to the designing. (2) To validate the efficiency of the algorithms. Because in our framework, wsdl4j is used to parse the WSDL documents, and dom4j is used to parse BPEL documents. So, we now focus on the validation efficiency of our proposed algorithms. The validation program is completed with Java on the platform Eclipse.

We adopt the dataset from [14]. There are 351 available services which use 2891 parameters in their input and output messages. This dataset has four groups, where Group 1 and Group 2 are chosen in our experiment. Group 1 contains solutions with 9 levels and Group 2 contains solutions with 18 levels. In our experiments, a random service of every level is presumed to be failed. At each experiment, we run the validation algorithms and running time is recorded. At Last, each data point is obtained from the average of three runs for the different failed service.

For the validation of fault-handling shown in Fig. 7, we change the size of the set *Wsf* of the failed service. Overall, the maximum running time is less than three milliseconds even though we set the size 100. Comparing the Group 1 with Group 2, there is not quite a difference in the running time in spite the fact that the levels of Group 2 is twice as many as the levels of Group 1. The main source of this conclusion is that the running time does not depend on the level of the service but the size of the set *Wsf* of the failed service. For designers, the running efficiency is quite acceptable.

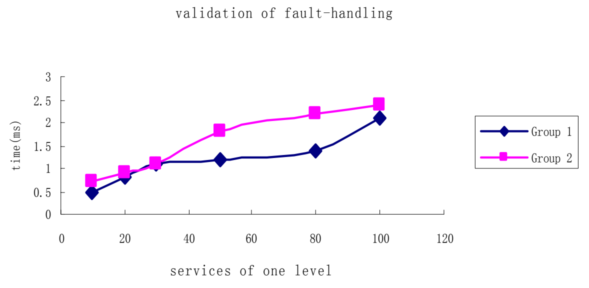
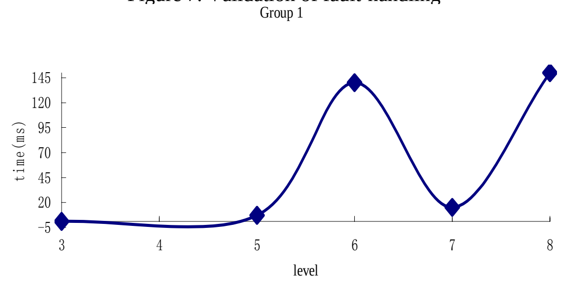
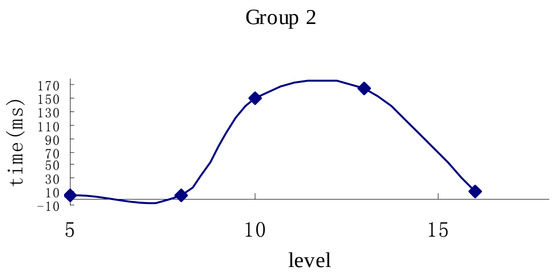


Figure 7. Validation of fault-handling



(a) Validation of Group 1



(b) Validation of Group 2

Figure 8. Validation of compensation-handling

For the validation of compensation-handling, because of their different levels, we place the running results of Group 1 and Group 2 into two figures, i.e., Fig. 8.a and Fig. 8.b. respectively. In the case of several paths existence from the faulty service to a consistent service, the running efficiency is very excellent. For example, the running time is only about 10 milliseconds even though the faulty service is in the tenth layer in the Fig. 8.b. But, it is very time-consuming to find a solution according to the del information of the faulty service. For example, the running time reaches 164 milliseconds in the thirteenth layer. At the same time, we can observe that it will take more time to make the validation when the faulty service is in the later layer under the same conditions. For example, the time taken in the thirteenth layer is longer than the tenth layer.

From the above analysis, it is feasible to take use of our method and it is acceptable for the designers.

VI. CONCLUSION

In this paper, we focused on the validation of the correctness of the service composition process during design phase. To improve the fault-handling mechanism, we extend the BPEL with two operators, whose semantics are presented. Then the graph planning technology is introduced to validate the fault-handling and compensation-handling mechanisms. The algorithms are detailed respectively and the validation framework is described. The experiment is implemented and the results show that our proposed approach is effective.

For the operator *retry*, we only consider that one service is replaced with another. But, in actual application, one service may be replaced by several services which are combined to satisfy the functional requirement. The loop structure is also not considered in our current solution. Theses will be discussed in our further study. It is limited to guarantee the composition running successfully only with the validation during the design phase in the dynamic environment. So, another part of our future work is to integrate our approach into a self-adaptive framework which can monitor the process execution.

ACKNOWLEDGEMENT

This work is funded by the Natural Science Foundation of China (NSFC, Grant No.60873237), Natural Science Foundation of Beijing (Grant No.4092037), and partially supported by Beijing Key Discipline Program.

REFERENCES

[1] A. A. A. Alves, S. Askary, and et al. April 2007, *OASIS Standard Web Services Business Process Execution Language Version 2.0.* . Available: http://docs.oasis-open.org/wsbpel/2.0/serviceref, [retrieved: March, 2012].

[2] F. Leymann. May 2001, *WSFL: Web Serices Flow Languag*. Available: http://xml.coverpages.org/WSFL-Guide-200110.pdf, [retrieved: March, 2012].

[3] S. Thatte. June 2001, *XLANG: Web Service for Business Process Design.* Available: http://xml.coverpages.org/XLANG-C-200106.html, [retrieved: March, 2012].

[4] M. B. a. C. Ferreira., "An operational semantics for stac,a language for modelling long-running business transactions.," in *Proceedings of Sixth International Conference on Coordination Models and Languages,*, February 2004, pp. 87-104.

[5] Z. Qiu, S. Wang, G. Pu, and X. Zhao, "Semantics of BPEL4WS-Like Fault and Compensation Handling," in *FM 2005: Formal Methods*, ed, 2005, pp. 350-365.

[6] L. Chenguang, Q. Shengchao, and Q. Zongyan, "Verifying BPEL-Like Programs with Hoare Logic," in *TASE '08. 2nd IFIP/IEEE International Symposium on Theoretical Aspects of Software Engineering* 2008, pp. 151-158.

[7] Z. Huibiao, H. Jifeng, L. Jing, and J. P. Bowen, "Algebraic Approach to Linking the Semantics of Web Services," in *SEFM 2007. Fifth IEEE International Conference on Software Engineering and Formal Methods*, 2007, pp. 315-328.

[8] D. Ziyang, A. Bernstein, P. Lewis, and L. Shiyong, "Semantics based verification and synthesis of BPEL4WS abstract processes," in *Proceedings. IEEE International Conference on Web Services*, 2004, pp. 734-737.

[9] X. Fu, T. Bultan, and J. Su, "Analysis of interacting BPEL web services," in *Proceedings of the 13th international conference on World Wide Web*, New York, NY, USA, 2004, pp. 621-630.

[10] M. Xiaoyong, F. Yiyan, W. Yonglin, and F. Xia, "A petri net-based failure handling model for composition transactions," in *Second International Conference onComputational Intelligence and Natural Computing Proceedings (CINC)*, 2010, pp. 378-381.

[11] G. Friedrich, M. Fugini, E. Mussi, B. Pernici, and G. Tagni, "Exception Handling for Repair in Service-Based Processes," *IEEE Transactions on Software Engineering,* vol. 36, pp. 198-215, 2010.

[12] W. Gaaloul, S. Bhiri, and M. Rouached, "Event-Based Design and Runtime Verification of Composite Service Transactional Behavior," *IEEE Transactions on Services Computing,* vol. 3, pp. 32-45, 2010.

[13] Q. L. An Liu, Liusheng Huang,Mingjun, Xiao, "FACTS: A Framework for Fault-Tolerant Composition of Transactional Web Services," in *IEEE Transactions on Services Computing*, 2010, pp. 46-59.

[14] Y. Yan, P. Poizat, and L. Zhao, "Self-Adaptive Service Composition Through Graphplan Repair," in *IEEE International Conference on Web Services (ICWS)*, 2010, pp. 624-627.

[15] C. Tien-Dung, P. Felix, and R. Castanet, "WSOTF: An Automatic Testing Tool for Web Services Composition," in *Fifth International Conference on Internet and Web Applications and Services*, 2010, pp. 7-12.

[16] E. Martinez, M. E. Cambronero, G. Diaz, and V. Valero, "Design and Verification of Web Services Compositions," in *International Conference on Internet and Web Applications and Services* 2009, pp. 395-400.

[17] C. Tien-Dung, R. Castanet, P. Felix, and G. Morales, "Testing of Web Services: Tools and Experiments," in *IEEE Asia-Pacific Services Computing Conference (APSCC)*, 2011, pp. 78-85.

[18] M. L. F. Avrim L. Blum, "Fast Planning Through Planning Graph Analysis," in *Proceedings of the International Joint Conference on Artificial Intelligence(IJCAI)*, 1995, pp. 1636-1642.