International Journal of Signal Processing, Image Processing and Pattern Recognition
Vol.8, No.7 (2015), pp.341-350
http://dx.doi.org/10.14257/ijsip.2015.8.7.33

A New Formal Modeling Method for Web Service Composition

Lin Li[1,2] and Cheng Wang[3]

[1] College of Computer Science and Technology, Wuhan University of Science and Technology, Wuhan, Hubei, China 430000
[2] Hubei Province Key Laboratory of Intelligent Information Processing and Real-time Industrial System, Wuhan, Hubei, China 430000
[3] Department of Common Required Courses, Hubei Institute of Fine Arts, China
lilin@wust.edu.cn, wangchengwh@qq.com

Abstract

With the rapid development of e-commerce, the application of web service composition is becoming increasingly important in many areas. If we apply unchecked web service composition, then we can only discover problems afterwards, and we have to devote unnecessary energy to repair. Here, formal modeling serves to be a premise to check. This paper brings up a formal modeling method for web service composition based on BDL (behavioral description language). The structure of this essay is to analyze the description of BPEL process, and we can get the formal model from the BPEL-to-BDL mapping relation.

Keywords: *Web service composition, formal modeling, behavior-oriented modeling language*

1. Introduction

Formal methods [1] have rigorous foundation of mathematics, and can describe and test software systems accurately, so they can help software developers to understand the system, to find inconsistent, inaccurate or inadequate descriptions of the system so as to correct mistakes and flaws of the design. So formal methods are important to guarantee the correctness of software and to improve the safety and credibility of software system. Thus, it's a hot issue to use formal methods, such as process algebra [2,3] and Petri-net [4], to describe and test web service composition.

This essay is about web service composition--the hot issue which has profound technological background and wide application prospect. This essay talks about the modeling method based on BDL, the mapping from BPEL4WS to BDL and modeling method. I will first establish a mapping relation between BPEL4WS and BDL, and then conduct model transformation of the atomic behaviors, structural behaviors and parallel behaviors of BPEL4WS. After that, we can get a model based on software behaviors by using BDL. This builds bases for future formalized verification work.

2. Behavior Description Language

Problem domain was divided into several sub-problem domains by using the method, and a requirement model for the system was established based on different users' viewpoints. Meanwhile, the important characteristics such as information security of complex system were checked by using semantic model and temporal logic of actions. Behavior description language (BDL) and syntax tree were mainly used in the construction of behavior model, and the semantic model for the behavior model was built by applying operational semantics to test the behavior model. Considering BDL [5], it reflects the behavior of the system using behavior expressions and explains the state change of the system using the variation of the behavior expression. Besides, BDL shows the behavior trace of the system as well.

A behavior is a certain interaction among two or more entities. For easy discussion, this paper presumes a behavior is an interaction only between two entities. We define a software behavior as a process during which a subject implements an operation, service, or action to an object. The subject and the object which may be physical or logistic, can be a person, a software or hardware component of system, or certain element of environment.

The structure of each behavior consists of a subject, an object, some properties, some inputs, some outputs, and an operation, service, or action. If a behavior can't be divided into two or more sub-behaviors, it is an atomic behavior. An atomic behavior is a simple behavior. Two or more simple behaviors form a composite behavior. In addition, according with the interact mode of software behaviors, the combine pattern of simple behaviors can be divided into five categories: sequence, certainty choice, uncertainty choice, parallel and shielding.

Suppose ABehID, ABehIDi (i∈ N) are atomic behavior identifier, BehID, BehIDi (i∈ N) are behavior identifier.

A. Atomic Behavior Expression

ABehID: f (sub, obj[& notes about obj]) [(x1, …, xn)]
[When prepositive conditions]
[InFrom (IID) (u1, …, un)]
[OutTo (OID) (v1, …, vm)].

where f is a service, an operation or an action. The subject sub commits f to obj; x1, x2, …, xn are variables; InFrom clause means this behavior takes input data u1, u2, …, un from IID. IID could be an identifier of another behavior, IO Pool or external entity. OutTo clause means this behavior provides output data v1, v2, …, vm to OID;

1)Idle behavior

ABehID: Idle //Idle behavior means nothing occurs.

2)Return behavior

ABehID: Return () //A return behavior is used when a composite behavior is about to exist or jump to a specific behavior.

B. Composite behaviors and behavior expressions

Let α-denotes that "the string α is a valid behavior expression". We can define behavior expressions as follows:

a) The identifier of an atom behavior is an expression.

|— ABehID

b) Sequence behavior:

1) $\dfrac{|-ABehID_1 \& |-ABehID_2}{|-ABehID_1; ABehID_2}$ 2) $\dfrac{|-ABehID \& |-BehID_1}{|-ABehID; BehID_1}$

3) $\dfrac{|-BehID_1 \& |-ABehID}{|-BehID_1; ABehID}$ 4) $\dfrac{|-BehID_1 \& |-BehID_2 \& ..... \& |-BehID_n}{|-BehID_1; BehID_2; ...; BehID_n}$

c) Conditional behaviors：

$\dfrac{|-BehID_1 \& |-BehID_2 \& b}{|-If\ b\ Then\ BehID_1\ Else\ BehID_2\ Fi}$, where b is a Boolean expression.

d) Choice behaviors

$\dfrac{|-BehID_1 \& |-BehID_2 \& ..... \& |-BehID_n}{|-BehID_1 + BehID_2 + ... + BehID_n}$

e) Parallel expressions

$\dfrac{|-BehID_1 \& |-BehID_2 \& ..... \& |-BehID_n}{|-BehID_1 \| BehID_2 \| ... \| BehID_n}$

3. BPEL Atomic Behaviors Modeling

The basic activity of BPEL4WS is to execute the interactive activities of WS. Interactions all include the partnerLink, portType and operation of two partners.

1. <Receive> Activity

Receive activity has to wait for an invocation request and is described as follows:

```
<process name="pname" ...>
  < receive
      partnerLink="tdname"  portType="qname"  operation="tdname"
      variable="value" >
  < /receive >
  ...
```

Prior to the Receive activity, a process name is added to explain the progress name of Receive activity. Variable in Receive activity is designed to receive request parameters, and *partnerLink, portType and operation* are the service access points of the invoker. The behavior therefore can be modeled as: requesters soc=Source(*partnerLink, portType and operation*) waiting for requests receive request parameters through chs=Channel(*partnerLink, portType and operation*), and a passage name IOport for returning results in the future (if necessary). By describing using BDL, the model is:

$| \mathit{receive}_{BP}|_{BDL} = \mathit{receive(soc, pname)}\ \ \mathit{Infrom\ (soc)(value)}$

2. <Reply> activity

Correspondingly, Reply behavior returns a reply, as explained below:

```
<process name="pname" ...>
...
   <reply
      partnerLink="tdname" portType="qname" operation="tdname"
      variable="value" >
   </reply>
```

A *process name* is added prior to the *Reply* behavior to show the progress name of the behavior. Reply results are assigned by *variable* in Reply behavior, and *partnerLink, portType and operation* appoint the service access points of respondents accepting requests. A model is established for the behavior as follows: respondents return response to requester soc=Source(*partnerLink, portType and operation*) through a passage chs=Channel(*partnerLink, portType and operation*) and accept a request parameter and a passage name IOport for returning results in the future (if necessary). BDL behavior is presented as:

$| \mathit{reply}_{BP}|_{BDL} = \mathit{reply(pname, soc)}\ \ \mathit{Outto(soc)(value)}$

3. <Assign> activity

Assign activity is described as follows in a web service:

```
   < assign >
     < copy >
       < from variable="value" part="AData">
       < to variable="vname"  part="AData" >
     < /copy >
   < /assign>
```

The Value is assigned to a variable vname in Assign behavior. In the modeling using BDL, value is transmitted through passage ASport. To ensure the security of the passage, the passage is defined as private to prevent the transmission of value from outside interference. BDL behavior is presented as:

$| \mathit{assign}_{BP}|_{BDL} = \mathit{assign(Subject1,Object1)}$
$\qquad \mathit{Infrom\ (AData)(value)}\ \ \mathit{Outto\ (AData)(vname)}$

4.<Invoke> activity

Invoke activity is described as follows:

```
<invoke
  partnerLink="ncname" portType="qname" operation="ncname"
  inputVariable="request" outputVariable="response" >
</invoke>
```

In Invoke behavior, *inputVariable* appoints the invoking input parameter, *outputVariable* is designed to return the invoking results, and *partnerLink*, *portType* and *operation* are the service access points to be invoked. The behavior is modeled as: invokers send input variables through passage chs=Channel(*partnerLink, portType and operation*), and invokers send a request and receive a response. When used in BPEL, there are two inter-behaviors in Invoke behavior. Therefore, two passages DataCell1 and DataCell2 are used in corresponding BDL behavior expression.

$| invoke_{BP}|_{BDL} = invoke(Sub1,ncname)\ Infrom(Sub1)(request)$
$returnvk（ncname，Obj1）Outto(Obj1)(response)$

5. <Throw> activity

Throw activity has to throw an exception and is described as follows:

```
<throw>
  faultName="fname"  faultVariable="fparam" >
</throw>
```

Regarding the exception handling of BPEL, each exception presents a unique name and relevant exception parameter, *faultVariable*. Exception processor catches exceptions based on the names of exceptions. Therefore, to simulate the exception throwing mechanism, a specific passage name, *faultport*, has to be formulated to describe the exception throwing processing. In the model, Throw behavior is similar to Reply behavior, while the former applies specific passage name, which is in favor of the identification of the progress and the catch of exceptions by the processor. A model for throwing behavior is constructed using BDL as follows:

$| throw_{BP}|_{BDL} = throw(fname,system)\ Outto(system)(fparam)$

6.<Empty> activity

Empty behavior represents a behavior without any action, so $| Empty_{BP} |_{BDL} = idle$.

4. BPEL Structural Behaviors Modeling

1. <Pick> activity

Pick activity means that after one event in an event collection occurs, it would interact with the occurred event. Each pick should include more than one *onMessage*. Pick activity can be described as follows with *variable* referring to the parameter of the event:

```
< Pick...>
  < onMessage...variable="event1">
    < activityl>
  < /onMessage>
  < onMessage，variable="event2">
    <actibity2>
  < /onMessage>
< /Pick>
```

The semantics of *onMessage* is totally same with that of the re behavior in above pick definitions, and can be used as input events in the BDL based modeling. Parameters *event1* and *event2* are events in the event set, and *activityl* and *activity2* represent actions executed after *event1* and *event2*. In BDL, "+" shows the uncertain optional relation, and the behaviors of each sub-division present a prefixion. When corresponding output prefixion appears in BDL, the sub-division is optionally executed.

By using BDL, the process of the behavior is described as:

$| Pick_{BP}|_{BDL} = receive(...)\ Infrom\ (soc)(value)$ ；

```
    activity1 （...） When （value=event1 ）
  + activity2 （...） When （value=event2 ）
```

2. <Sequence> activity

Sequence activity defines a set of sub-behavior executed in orders, and it can be described as follows:

```
< sequence>
  <actl>
  (act2)
   ...
  <actn>
< /sequence>
```

This means that after activity 1 (actl) is conducted, act2 follows, and until actn. Actl, act2 and actn make up a sequence being conducted in orders. Se activity of BPEL corresponds to the * actor of BDL to mean sequential operation. In BDL, sequence activity is expressed like this:

$| Sequence_{\mathrm{BP}}|_{\mathrm{BDL}} = |act1_{\mathrm{BP}}|_{\mathrm{BDL}} ; |act2_{\mathrm{BP}}|_{\mathrm{BDL}} ; \ldots |actn_{\mathrm{BP}}|_{\mathrm{BDL}}$

$= ABeh1 ; ABeh2 ; \ldots ABehn$

3.<Switch> activity

Switch activity chooses one activity from several sub-behaviors to conduct based on one or more selection criteria branch.

```
<switch >
   <case condition="bool-expression">
     <activity1>
     </activity1>
   </case>
   <otherwise>
      <activity2>
      </activity2>
   </otherwise>
</switch>
```

Switch has one criterion or many criteria, thus defining one or more cases. The condition of case is the boolean expression of certain variables. Each case corresponds to an activity which may be executed. We build our model on BPEL under condition 2, because more complicated cases can be nested under condition 2. Switch activity of BPEL means that Switch chooses one behavior among many sub-behaviors to conduct based on one or more selection criteria branch. In BDL, we can map by using a standard process expression and alternative choices:

```
| Switch_BP |_BDL = if(bool-expression=case1) then activityl(...)
          else if(bool-expression=case2) then activity2(...)  fi
```

4.<While> activity

While activity allows sub-behaviors to be conducted many times until curtain criteria can't be met.

```
<while condition="bool-expression" >
    <activity>
    </activity>
</while>
```

In the behavior, when the condition *bool-expression* is satisfied, sub-behavior ac has to be continuously executed. The interative behavior has to be defined using the recursion of the behavior in BDL.

```
| While_BP |_BDL = AbehID:  activity(Subject,Object)
                  When precondition= bool-expression
                  [ Infrom (SourceID) (u1,u2,...un)]
                  [ Outto (TargetID) (v1,v2,...vn)]
```

Returnto (ABehID)

The *Returnto* here and the following mark *ABehID* constitute a recursive definition, which describes that if the pre-condition *bool-expression* is "true", *activity* is executed; if the condition is still "true" afterward, *activity* is executed again; the behavior is continuously performed until the pre-condition is "false".

5. <flow> activity

Flow behavior defines a group of sub-behaviors which are concurrently executed.

```
<flow>
   < activityl >
   < /activityl >
   < activity2 >
   < /activity2 >
< /flow >
```

There are two sub-behaviors, ac1 and ac2, that concurrently executed in the given Flow behavior. The concurrent operation of the process in BDL is realized by composition operator " | ".

$$| Flow_{BP} |_{BDL} = |activity1_{BP} |_{BDL} \ | \ | activity\ 2_{BP} |_{BDL}$$
$$= ABeh1|ABeh2$$

5. Case Study

Now, we will use a composite service described by BPEL as an example to show executing procedure of the algorithm. Composite service is described by nested structure and concurrent behavior. The composite service used here is a booking process: when the system receives an order from the client, the service system would have two concurrent sub-missions, i.e., to arrange the train and to calculate price, when the two missions are completed, the ticket will be given to the client. This composite service is described by BPEL like this:

```
------------------------------------------------------------------------------------------------------------------
----- <process name="purchaseTicketProcess" ...>
  <sequence>
     <receive partnerLink="purchasing"  variable="PT" .../>
     <flow>
       <sequence>
         <assign>
           <copy>
             <from variable="PT" .../>
             <to variable="TrainRequest" .../>
           </copy>
         </assign>
         <invoke partnerLink="Train"
           inputVariable=" TrainRequest"
           outputVariable="TrainSchedule " .../>
      </sequence>
      <sequence>
        <assign>
          <copy>
            <from variable="PT" .../>
            <to variable="ticketRequest" .../>
          </copy>
        </assign>
        <invoke partnerLink="ticketing"
          InputVariable=" ticketRequest "
          outputVariable=" ticket " .../>
     </sequence>
  </flow>
```

```
   <reply partnerLink="purchasing" variable="ticket" .../>
  <sequence/>
 </process>
```

Figure 1. BPEL Description of the Composite Service

Via syntax tree node naming rules, we convert the BPEL process of booking example into arborescent structure. The arborescent structure is as shown in Figure 2 below.

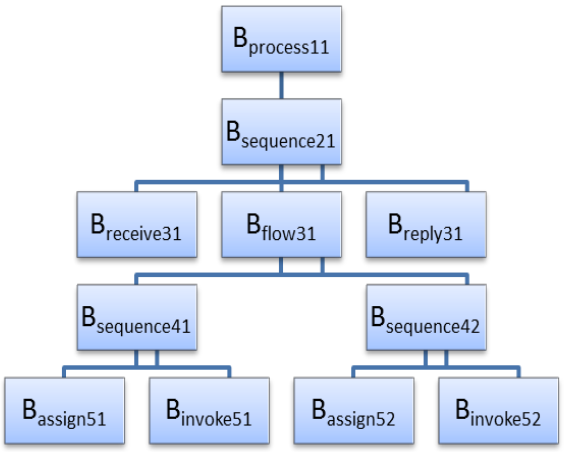


Figure 2. The Arborescent Structure of Booking Example

This implementation iterates over the specified syntax tree, and gets the behavioral trace, which is shown as follows:

Bprocess11 = Breceive31->
Bsequence21->
Bassign51->Bsequence41->Binvoke51->
Bflow31->
Bassign52->Bsequence42->Binvoke52->
Bsequence21->
Breply31

Symbolic representation of the behavior sequence is got by using the mapping rules of structural mapping library:

Bprocess11 = Breceive31 ; (Bassign51 ; Binvoke51) | (Bassign52 ; Binvoke52) ; Breply31

And then , by using the mapping rules of atomic mapping library, we convert the nodes to the corresponding atomic behavior sequence:

Breceive31 = receive(client, PurchaseTicketprocess) Infrom (client)(PT)
Bassign51 = assign(Subject1, Object1)
Infrom (Datacell1)(PT) Outto (Datacell1)(TrainRequest)
Binvoke51 = invoke(Sub2, Train) Infrom(Sub2)(Trainrequest)
returnvk （Train, Obj2） Outto(Obj2)(TrainSchedule)
Bassign52 = assign(Subject1, Object2)
Infrom (Datacell2)(PT) Outto (Datacell2)(TicketRequest)
Binvoke52 = invoke(Sub3, ticketing) Infrom(Sub3)(ticketRequest)
returnvk （ticketing, Obj3） Outto(Obj3)(ticket)
Breply31 = reply(PurchaseTicketprocess, client) Outto(client)(ticket)

Eventually, the complete behavior description language sequence is as follows:

PurchaseTicketProcess = receive(client, PurchaseTicketprocess) Infrom (client)(PT) ;
(assign(Subject1, Object1)
Infrom (Datacell1)(PT) Outto (Datacell1)(TrainRequest) ;

```
invoke(Sub2, Train)  Infrom(Sub2)(Trainrequest)
  returnvk （Train, Obj2）  Outto(Obj2)(TrainSchedule)
) |
(assign(Subject1, Object2)
  Infrom (Datacell2)(PT)   Outto (Datacell2)(TicketRequest) ;
invoke(Sub3, ticketing)  Infrom(Sub3)(ticketRequest)
  returnvk （ticketing, Obj3）  Outto(Obj3)(ticket)
) ;
reply(PurchaseTicketprocess, client)  Outto(client)(ticket)
```

6. Conclusion

This essay brings up a new formal modeling method for Web service composition based on BDL, this method is the premise to check and the base for future work. This modeling method first analyzes the given BPEL process, and gets BDL behavioral syntax tree through node naming rule of syntax tree, then produces trace behavior through inorder traversal, and finally gets the corresponding formal model through the mapping relation from BPEL to BDL. This essay introduces the concept mapping between BPEL to BDL and the model transformation between atomic behaviors, structural behaviors and parallel behaviors. The mapping relation between BPEL and BDL is vital, and the mapping rule is like this: we first establish a mapping relation between the conception of BPEL and BDL: a web service composition business process described by BPEL4WS corresponds to a BDL viewpoint; a web service defined by BPEL4WS corresponds to a BDL scenario; a finest behavior defined by BPEL corresponds to a BDL atomic behavior; inner-communication between BDL behaviors is used to describe the interactive relationship between web services; variables in BPEL can be reflected as the exchanged information of BDL behavioral communication; the partner Link, port Type and operation of BPEL describe the interactive access point of web service, and correspond to the IO channel used by BDL behavioral communication. Then we establish model transformation between the atomic behaviors receive, reply, assign, invoke, throw, empty, and structural behaviors pick, sequence, switch, while of BPEL on the one hand, and parallel behaviors flow on the other. Then, this essay analyzes the whole process of formal modeling through cases, which is significant for other researchers' future work. In the future, we will test related properties of Web Service composition based on this essay.

Acknowledgements

The research work was supported by Wuhan University of Science and Technology Youth Training Plan (2014xz017), the National Natural Science Foundation of China (Grant Nos. 61273225, 61472293, 61440016), the Natural Science Foundation of Hubei Provincial of China (2014CFB247).

The corresponding author is Cheng Wang.

References

[1] M. Hinchey, M. Jackson, P. Cousot, B. Cook, J. P. Bowen and T. Margaria, "Software engineering and formal methods. Communications of the ACM, vol. 51, no. 9, **(2008),** pp. 54-59.

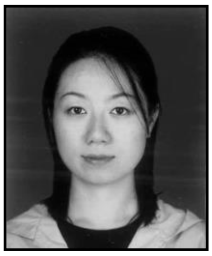
[2] J. C. Baeten, T. Basten, T. Basten and M. A. Reniers, "Process algebra: equational theories of communicating processes", Cambridge university press, vol. 50**, (2010).**

[3] M. E. Cambronero, G. Díaz, V. Valero and E. Martínez, "Validation and verification of Web services choreographies by using timed automata", The Journal of Logic and Algebraic Programming, vol. 80, no. 1, **(2011),** pp. 25-49.

[4] P. Xiong, Y. Fan and M. Zhou, "A Petri net approach to analysis and composition of web services", Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on, vol. 40, no. 2, **(2010),** pp. 376-387.

[5] L. Li, G. Wu, H. Bo, W. Li and W. Hao, "Behavioral Model Based Requirements Visualization Method", Chinese Journal of Computers, vol. 36, no. 6, **(2013),** pp.1312-1324.

Authors



Li Lin, she received M.Sc. in computer applications (2006). Since 2010 she becomes Ph.D. candidate in computer software and theory from Wuhan university. Since 2009 she is lecturer of Wuhan University of science and technology. Her research interests include requirements engineering and formal method and requirements visualization.

Wang Cheng, he received M.Sc. in computer applications (2006). Since 2009 he is lecturer of Hubei Institute of Fine Arts. His research interests include Computer Graphics and Image Processing and Virtual Reality.