2010 Fifth International Conference on Internet and Web Applications and Services

The Research on Formal Specification and Review of Workflows

Quanmin Fu, Jian Liang, Haopeng Chen, Fenglin Bu
Shanghai Jiao Tong University
Shanghai, P.R. China
fuquanmin@yahoo.cn, tianlankong@yahoo.com.cn, chen-hp@sjtu.edu.cn, bu-fl@cs.sjtu.edu.cn

***Abstract*—The workflow technique based on BPEL is widely used in the management of enterprise business process. But it is hard to detect logic errors and conduct the validation of the business process, because BPEL does not have formal semantics. This paper introduces SOFL to describe business process. Generally speaking, SOFL provides a formal but comprehensible language for both requirements and design specifications, and a practical method for developing software systems. This paper extends SOFL to describe BPEL workflows and propose basic rules to transform BPEL into SOFL to demonstrate the ability of describing of SOFL. Then, an example is given to illustrate the rules. After that, this paper shows how to conduct the formal review of the SOFL module which is transformed from BPEL process, to ensure the internal consistency and validity of the business process.**

Keywords-BPEL; SOFL; Formal Specification; Formal Review

I. INTRODUCTION

Workflow is the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules. Using computer to assist the coordination of business process can improve the efficiency and the production capability. BPEL is now the most widely used language to describe workflow.

Business Process Execution Language for Web Services (BPEL or BPEL4WS) is a language used for the definition and execution of business processes using Web services. BPEL provides a relatively easy and straightforward way to compose several Web services into new composite services called business processes. Though many BPEL design tools can help to find the syntax errors in the BPEL business process, due to the complexity of the business logic, the potential logic errors are still hard to be eliminated completely. Besides, WS-BPEL comes without a formal semantics and its specification document [1], written in 'natural' language, contains a fair number of acknowledged ambiguous aspects that may lead to different interpretations, e.g., the relationship between WS-BPEL (multiple) start activities and the mechanisms handling race conditions has not been fully explored; moreover, if suitable measures for 'protecting' such critical activities as fault and compensation handlers are not taken into account, then subtle behaviors can arise when implementing activities that cause immediate termination of other activities. Furthermore, the design of WS-BPEL applications is difficult and error-prone due to the presence of such intricate features as concurrency and race conditions, forced termination, multiple instances and message correlation, long-running business transactions and compensation handlers. It would thus benefit from the use of formal methods because these can provide a framework to precisely describe some aspects of an application, to state and prove its properties which contain internal logic and whether it satisfies the requirement of the users, and to direct attention towards issues that might otherwise be overlooked, such as whether there are dead locks or unviable activities.

Formal method has advantages of precise description and verification. SOFL [2] is chosen as transforming target because SOFL is structured, object-oriented and formal. It integrates data flow diagrams, Petri nets and the object-oriented approach in a coherent manner for specifications construction, and also integrates formal verification [3] with fault tree analysis [4] and testing for specifications [5]. Furthermore, with tool support, the animation of a SOFL specification can assist its user to review the adequacy of the specification [6]. Currently SOFL just provides a rigorous, systematic and effective software developing method to describe requirements and design specifications, but does not have close relation to Web service technology and SOA architecture.

This paper introduced a method for transforming BPEL into SOFL. The purpose is to merge the merits of both proposals, and to describe business process for depicting workflow more explicitly.

The rest of this paper is organized as follows. The next section provides a summary of related work. Section 3 first makes a brief introduction of workflow technology, and then describes the basic rules of transforming BPEL into SOFL. Section 4 gives an example of transforming. Section 5 describes how to conduct the formal review of the SOFL module which is transformed from a BPEL process. Finally, concluding remarks are drawn in Section 6 with some directions for future work.

II. RELATED WORK

Because BPEL lacks of a formal semantics and contains ambiguities, several attempts have been made to formalize BPEL [7] these years, using automata [8], process algebra [9], Petri nets [10], etc.

Automata is a public and base model of formal specifications for systems. In the reference [11], Diaz et al. convert automatically business processes written in BPEL-WSCDL to timed automata. The tool developed in the reference [12] accepts BPEL specifications as input and translate them to guarded automata. But some advanced features in BPEL, e.g., the endpoint references (to dynamically determine the peer to talk to), cannot be

978-0-7695-4022-1/10 $26.00 © 2010 IEEE
DOI 10.1109/ICIW.2010.11

captured in that model.

The semantic foundation of process algebras is based on automata [13]. In the paper [14], Ferrara defines correspondence between BPEL and LOTOS, and gives guidelines for translations between a process algebra and BPEL. The advantage of this proposal is that it includes compensations and exception handling, but do not consider dynamic process instantiation and correlation set.

Petri nets provide a broad basis for computer aided verification. Currently, transforming BPEL into Petri nets is a popular method of formalizing business process [15]. The paper [16] proposes a formal Petri nets semantics for BPEL which assures exception handling and compensations. In the paper [17], Yang et al. present the transformation of BPEL to Colored Petri nets (CP-nets) in a constructive way. Yang et al. also make tool support available for verifying BPEL composition.

Petri nets are primarily used to provide an operational semantics for the data flow diagrams, while SOFL integrates Petri nets. The following is a brief comparison between Petri nets and SOFL:

1) PN is limited in describing complicated functions with rich data types and high logical complexity.

2) PN is a graphical notation and therefore its expressiveness is limited for large scale systems (e.g., the diagram is complex and inefficient for evolution).

3) SOFL strikes a balance between diagrammatical representation based on the intuitive data flow concept and rigorous textual expression using pre- and post- conditions. The formalized diagram, called condition data flow diagram (CDFD) is suitable for describing the architecture of the system, while the formal textual notation is effective in defining the meaning of its components, including processes and data flows.

4) A SOFL specification has a very good traceability for specification evolution and verifiability for checking its consistency and validity. It is also effective in instructing programmer to implement his or her programs, and to facilitate the testing of the program.

III. FORMAL SPECIFICATION OF WORKFLOW

Nowadays, workflow technology has been intensively applied in the domains of administration, production, and scientific research. The process definition is the core point of the workflow specification. The Workflow Management Coalition developed a meta-model for the process definition, which identifies a basic set of objects types appropriate to an initial level for the interchange of relatively simple process definitions. Further object types may be added, either by vendor specific extensions and/or by defining additional conformance levels with added functionality. Figure 1 illustrates the elements and their relationships in the meta-model.

Based on the meta-model, many methods have been proposed to describe the business process, such as Transaction Flow Diagram, Role Activity Diagram, Event Process Chains, UML Activity Diagram, Petri-nets and BPEL. Because of the strong description ability, the executable feature and the support of many large companies, BPEL is the most widely used business process description language. But as mentioned in the Section 1, there are many unsolvable problems to use BPEL, e.g., BPEL doesn't provide the mechanism to ensure the internal consistency and the validation of the process. To solve these problems, the process needs to be described formally. This paper introduces SOFL to describe business processes. To demonstrate the description ability of SOFL, the following part of this section will give the basic rules of transforming a BPEL process into a SOFL module.

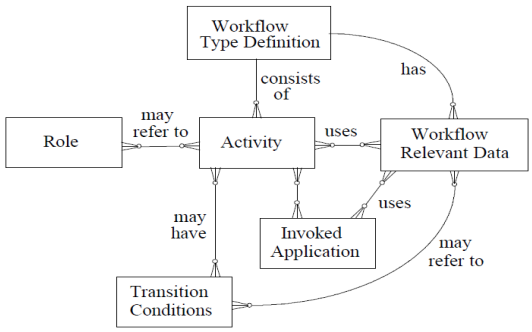

Figure 1. Basic Process Definition Meta-model

When a BPEL business process is described with SOFL, there are the following basic rules:

1. A BPEL process is transformed into a SOFL module

2. The basic activity in BPEL is transformed into the process in SOFL. Generally speaking, a basic activity is transformed into a process, but some can correspond to two or more processes(e.g., one <invoke> activity corresponds to two processes in SOFL).

3. The structured activity in BPEL is transformed into the combination of the structures and the processes in SOFL.

4. The message is the communication method in SOFL, so the message model in BPEL is easily transformed into SOFL.

5. When describing stateful business process in SOFL, the instance ID is embedded in the messages, the process instances accept the messages in which the instance IDs match. Through this way, the function of correlationsets in BPEL is realized.

6. The faultHandler and compensationHandler in BPEL can be transformed into the corresponding processes in SOFL. The processes corresponding to faultHandler are invoked by the processes transformed from the <throw> or <rethrow>. The processes corresponding to compensationHandler are invoked by the processes transformed from <compensate> or <compensateScope>.

7. Some activities (e.g., <foreach>) in BPEL are dynamic, their behaviors are determined at runtime. When the dynamic factors in these activities are determined, they become the same as the static activities. The dynamic activities can not be transformed into the existing elements in SOFL. This paper extends SOFL by defining the

dynamic structures. When the SOFL engine encounters these structures, it first determines the dynamic factors and then transforms them to static structures or processes.

The following subsections give the detail introduction of how to transform the basic activities and structured activities in BPEL.

A. Transforming Basic Activities

<invoke>

The <invoke> activity in BPEL is used to call Web Services offered by service providers. The typical use is invoking an operation on a service. Operations can be one-way or request-response operations.

A request-response invocation in BPEL can be transformed into two processes in SOFL:

1. A process to send invocation messages.
2. A process to receive return messages.

A one-way invocation in BPEL can be transformed into a SOFL process directly, which is the process to send invocation messages.

<receive>

The <receive> activity specifies the partner link it expects to receive from, and the port type and operation that it expects the partner to invoke.

A <receive> activity can be transformed into a process to receive messages. The process receives invocation messages as input messages.

<reply>

The <reply> activity is used to send a response to a request previously accepted through an inbound message activity such as the <receive> activity.

A <reply> activity can be transformed into a process to return messages in SOFL. The process sends the return messages to the process transformed from the corresponding <invoke> activity.

B. Transforming Structured Activities

<sequence>

The <sequence> activity is used to define a collection of activities to be performed sequentially in lexical order. The executing order in SOFL is determined by input and output. If the input of process B is the output of process A, then A is executed prior to B. Figure 2 illustrates the corresponding form of <sequence> in CDFD.

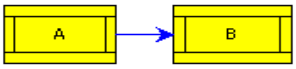

Figure 2. The sequence structure in SOFL

<if>

The <if> activity provides conditional behavior. The activity consists of an ordered list of one or more conditional branches defined by the <if> and optional <elseif> elements. SOFL provides a binary condition structure. Figure 3 illustrates the form of the <if> structure in CDFD. Out Activity in Figure 3 assigns the input to the output after any of its inputs arrives.

<while>

The <while> activity provides for repeated execution of a contained activity. The contained activity is executed as long as the Boolean <condition> evaluates to true at the beginning of each iteration. In SOFL, a binary condition structure with a judge process can be used to implement the <while> activity. Figure 4 illustrates the form of the implementation in CDFD. The Activity A here represents all the operations contained in the <while> activity, and maybe these operations are not executed.

<pick>

The <pick> activity is used to wait for one of several possible messages to arrive or for a time-out to occur. When one of these triggers occurs, it won't accept other triggers, and the associated child activity is performed. When the child activity completes then the <pick> activity completes.

Figure 5 illustrates the corresponding form of <pick> in CDFD. Every <onAlarm> activity in <pick> is transformed into an onAlarm process in SOFL. Every input port of the PICK process listens to a message or an alarm, when one of these triggers occur, the PICK process won't accept other triggers, and the corresponding processes are executed.

<flow>

The <flow> activity is used to specify one or more activities to be performed concurrently. The broadcasting structure can be used to specify the parallel processes in SOFL. The corresponding form of <flow> in CDFD is shown in Figure 6.

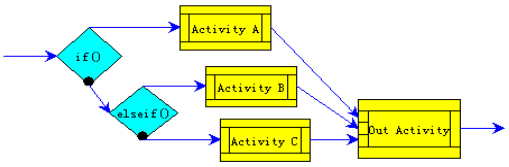

Figure 3. The corresponding structure of <if> in CDFD

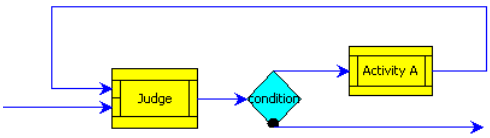

Figure 4. The corresponding structure of <while> in CDFD

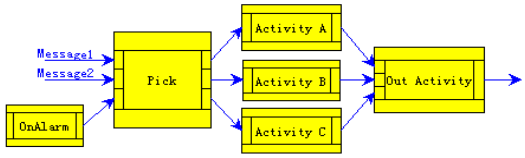

Figure 5. The corresponding structure of <pick> in SOFL

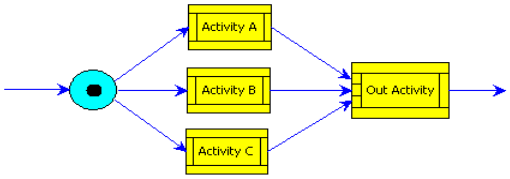

Figure 6. The corresponding form of <flow> in SOFL

IV. AN EXAMPLE OF TRANSFORMING BPEL TO SOFL

Let's consider a simplified business process for employee travel arrangements [18]: The client invokes the business process, specifying the name of the employee, the destination, the departure date, and the return date. The BPEL business process first checks the employee travel status, assuming that a Web service exists through which such checks can be made. Then the BPEL process will check the price for the flight ticket with two airlines: American Airlines and Delta Airlines. Again assume that both airline companies provide a Web service through which such checks can be made. Finally, the BPEL process will select the lower price and return the travel plan to the client.

Figure 7 shows a schematic view of the process.

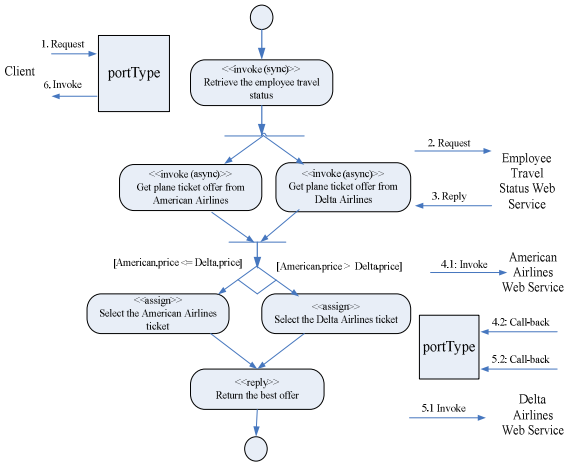

Figure 7. Example BPEL process for travel arrangements

Table Ⅰ. Data types of the example in SOFL specification

```
type
PortType = composed of
        module : String
        process : String
        port : String
        end
ClientInvokeMessage = composed of
                travelRequest : TravelRequest
                clientID : nat
                end
TravelStatusInvokeMessage = composed of
                    port : PortType
                    employee : Employee
                    end
ClientCallbackMessage = composed of
                  port : PortType
                  travelResponse : TravelResponse
                  end
TravelClass = {<economy>, <business>, <first>}
TravelRequest = composed of
            employee : Employee
            flightdata : FlightData
            end
TravelInfo = composed of
          travelclass : TravelClass
          flightdata : Flightdata
          end
FlightInfo = composed of
          price : real
          flight_no : string
          end
Employee = composed of
          name : string
          id : nat
          end
FlightData = composed of
           start_city : string
           end_city: string
           flighttime : FlightTime
           end
FlightTime = composed of
           time : Time
           date : Date
           end
Time = Hour * Minute * Second
Hour = nat0;
Minute = nat0;
Second = nat0;
Date = Day * Month * Year
Day = nat0;
Month= nat0;
Year = nat0;
TravelResponse = composed of
               flightinfo : FlightInfo
               company_name : string
               end
ResponseInfo = composed of
             travelResponse1 : TravelResponse
             travelResponse2 : TravelResponse
             end
```

This BPEL business process can be transformed into a SOFL module with CDFD and SOFL specification based on the rules of transforming in Section 3. An up-down scanning of the XML specification of BPEL can be done, and the activity can be transformed one by one. Finally, the CDFD of the business process for employee travel arrangements can be obtained, as shown in Figure 8. Table Ⅰ shows the data types of SOFL specification and Table Ⅱ shows part of corresponding SOFL specification details.

V. FORMAL REVIEW OF SOFL BUSINESSS PROCESS

Conducting formal review can ensure the validity of business process and reduce the cost and risk of software projects. Furthermore, for the business processes described by formal languages, more rigorous review techniques can be applied. To make reviews effective, especially for large-scaled and complex systems, it is important to use a systematic formal review method.

In the business process described by BPEL, little effort has been dedicated concerning the verification of the modeled business processes, e.g., there is no support to detect possible deadlocks, or to detect parts of the process that are not viable. Moreover, whether the business process reflects the user's requirements should also be concerned. If SOFL is used to describe the business process, the designers can take advantage of the SOFL review methods to verify and validate the business process. The review in SOFL can ensure not only the internal consistency of the business process, but also whether the business process satisfies the user's requirements. Both are hard to review in BPEL.

A. Review of the business process consistency

In order to ensure the internal consistency of business process, the designers should ensure that the CDFD output is generated consistently. This paper refers to the methods in paper [3], and proposes how to review the internal consistency of a business process. For the consistency of business process, each variable should be kept generated consistently. Suppose there is a process A with input int x, and output int y, and the process ahead of process A is process B. To ensure the output variable y generated consistently, the designers should review:

1) The output variable y of process A is in the postcondition of process A, i.e., y inset Variables(post_A).

2) Process A is satisfiable, i.e., forall[x : int] | pre_A (x)=> exists[y : int] | post_A(x , y). This means, for any input, if the precondition evaluates to true, there must exist an output based on which the postcondition evaluates to true.

3) Process A is consistent with the process ahead of it, i.e., pre_B and post_B(x) => pre_A

4) The input x of process A is generated consistently.

5) The output variable y of process A is viable, i.e., forall[x : int] | pre_A (x)=> exists[y : int] | post_A(x , y). This seems the same as (2). But they focus on different points. Step (2) focuses on process, while step (5) focuses on data flow. For the process which has several outputs, when at least one of the outputs is viable and the others

are unviable, the process is still satisfiable.

Then the variable x is reviewed. For the data flow of the business process, the formal review of the consistency is conducted from back to forward, until the input of the whole business process is met.

Assume there is a deadlock in the business process, e.g., the process C waits for the result of process D, and the process D also waits for the result of process C at the same time. When the reviews mentioned above start, suppose that the process C is reviewed first. The input of the process C is required, which is the output of the process D, generated consistently, and then the process D is reviewed. The input of process D is required, which is the output of the process C, generated consistently. So the process C have to be reviewed again, and it is obvious that a cyclic review is being conducted. A conclusion can be made that if cyclic review is detected, then a deadlock is found.

After all the previous reviews have been done, the feasibility and consistency of invariant will be reviewed to keep the SOFL business process internally consistent.

To review the invariant, two aspects must be checked:

1) The review of feasibility. For invariant constraint expression I: forall[x : D] | P(x). If I is feasible if and only if there exists an element r in D that P(r) holds.

2) The review of consistency. An invariant is consistent with the related process if it is not violated by those processes, before and after their executions. Let P be a process and I an invariant. I is consistent with P if and only if the following two conditions hold: pre_P and I <=> true and pre_P and post_P and I <=> true.

Paper [5] has more details of the review of invariant.

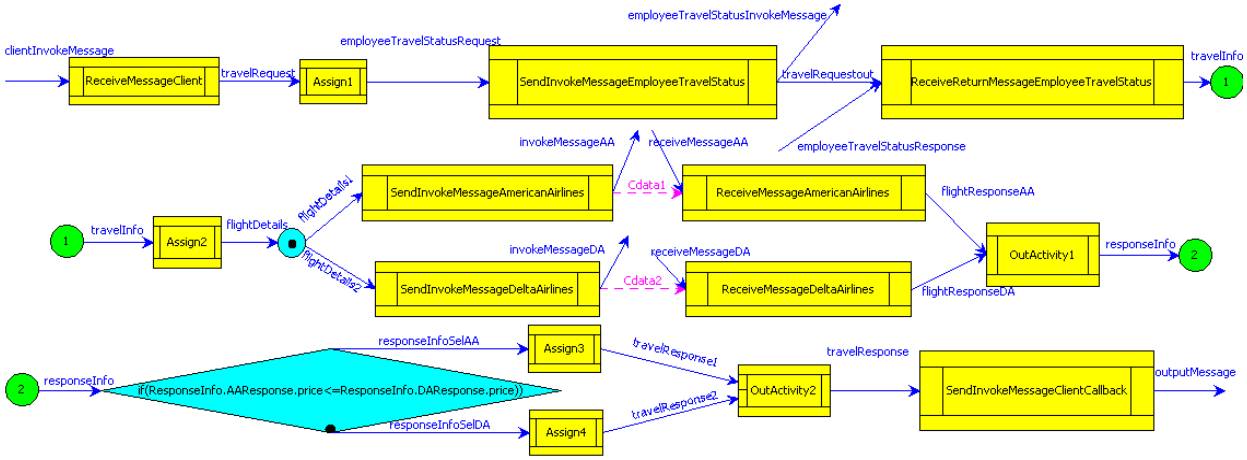

Figure 8. CDFD of the business process for employee travel arrangements

Table Ⅱ. Part of the SOFL specification of the business process for travel arrangements

```
process ReceiveMessageClient( clientInvokeMessage:ClientInvokeMessage ) travelRequest:TravelRequest
pre
   clientInvokeMessage <> nil
post
   travelRequest = clientInvokeMessage.travelRequest
end_process;

process Assign1( travelRequest:TravelRequest ) employeeTravelStatusRequest:TravelRequest
pre
   travelRequest <> nil
post
   employeeTravelStatusRequest.employee = travelRequest.employee
end_process;

process SendInvokeMessageEmployeeTravelStatus( employeeTravelStatusRequest:TravelRequest )
          employeeTravelStatusInvokeMessage:TravelStatusInvokeMessage, travelRequestout:TravelRequest
pre
   employeeTravelStatusRequest <> nil
post
   travelRequestout = employeeTravelStatusRequest and
   employeeTravelStatusInvokeMessage.port.module = EmployeeTravelStatusM and
   employeeTravelStatusInvokeMessage.port.process = EmployeeTravelStatusP and
   employeeTravelStatusInvokeMessage.port.port = EmployeeTravelStatusO
end_process;
```

```
process ReceiveReturnMessageEmployeeTravelStatus( employeeTravelStatusResponse:TravelClass,
                                                  travelRequestout:TravelRequest ) travelInfo:TravelInfo
pre
   travelRequestout <> nil and employeeTravelStatusResponse <> nil
post
   travelInfo.travelclass = employeeTravelStatusResponse and travelInfo.flightdata = travelRequestout.flightdata
end_process;

process OutActivity1( flightResponseAA:TravelResponse, flightResponseDA:TravelResponse )
                                                  responseInfo:ResponseInfo
pre
   flightResponseAA <> nil and flightResponseDA <> nil
post
   responseInfo.travelResponse1 = flightResponseAA and responseInfo.travelResponse2 = flightResponseDA
end_process;

process SendInvokeMessageClientCallback( travelResponse:TravelResponse )
                        outputMessage:ClientCallbackMessage
pre
   travelResponse <> nil
post
   outputMessage.travelResponse = travelResponse and
   outputMessage.port.module = ClientM and
   outputMessage.port.process = ClientCallbackPT and
   outputMessage.port.port = ClientCallbackO
end_process;
```

The internal logic of a SOFL business process can be reviewed by using the above method. In conclusion, the following properties can be reviewed:

1. The satisfiability of each process.
2. The viability of each data flow.
3. The consistency between two adjacent processes.
4. Whether the outputs of each process are in the post-condition, i.e., whether the process has invalid outputs.
5. Whether there are deadlocks in the process.
6. The feasibility and consistency of invariant.

B. Example:The review of business process consistency

Let us look back to the example made in the Section 4.

It is necessary to review the internal logic of the business process, in order to ensure the output outputMessage generated consistently from the input clientInvoke-Message through CDFD(Figure 8). The business process should be reviewed from back to forward, i.e., the consistency of outputMessage is reviewed first:

(1) The output variable outputMessage of process SendInvokeMessageClientCallback is in the postcondition of process SendInvokeMessageClientCallback.

(2) Process SendInvokeMessageClientCallback is satisfiable.

(3) The output variable outputMessage of process SendInvokeMessageClientCallback is viable.

(4) Process SendInvokeMessageClientCallback is consistent with the process ahead of it.

(5) The input variable of process SendInvokeMessage-ClientCallback travelResponse is generated consistently.

Then, the variable travelResponse should be ensured generated consistently:

(1) The output variable travelResponse of the process OutActivity2 is in the postcondition of OutActivity2.

(2) Process OutActivity2 is satisfiable.

(3) The output variable travelResponse of process OutActivity2 is viable.

(4) Process OutActivity2 is consistent with the process ahead of it.

(5) The inputs travelResponse1 and travelResponse2 of the process OutActivity2 are generated consistently.

After that, the variables travelResponse1 and travel-Response2 are reviewed respectively.

So the review will be conducted from back to forward until the input of the whole business process client-InvokeMessage, in order to ensure every variable generated consistently.

If the designers follow the above review procedures, review the SOFL business process from the output to the input, and never find that:

1. There exists a certain process which is unsatisfiable.
2. There exists a certain data flow which is unviable.
3. There exists a certain process which is not consistent with its adjacent process.
4. There exists a certain process in which its output variable is not in its postcondition.
5. There exists a deadlock in the business process.
6. The invariant is not feasible or consistent.

Then it can be judged that the internal logic of this business process described by SOFL is valid, and the internal consistency of the business process is ensured.

C. Review the validity of business process

To review the validity of the business process is to review whether the business process satisfies the requirements of the user. To do so, the designers should not just use some specific test cases to test whether the business process satisfies the requirements of the user, but logically review whether the outputs of the business process conform to the expected outputs under all the possible inputs. The designers of the business process can derive the expected pre- and postcondition of the business process based on the requirements of the user. As the process in SOFL is described with pre- and postcondition, the designers can deduce the pre- and postcondition of the whole business process from the pre- and postconditions of processes in it. Compare them with the expected pre- and postcondition, if they are equivalent, the business process is valid, i.e., it satisfies the requirements of user.

For the business process example given in Section 4, the designers can deduce the pre- and postcondition from the requirements of the user, as shown in Table Ⅲ.

In the above statement, the method getService(Input input, Service service) is used to invoke a service and it returns the callback message.

For the concrete SOFL business process, the designers can obtain the input of the whole business process travel-Request and the output of the whole business process travelResponse. Table Ⅳ shows the pre- and post-condition.

It is obvious that they are equivalent. So a conclusion can be made that this SOFL business satisfies the requirements of the user.

Table Ⅲ. The expected pre- and postcondition from the requirements of the user

```
ExpPrecondition: clientInvokeMessage <> nil

ExpPostcondition: flightDetails.travelClass = getSerive(clientInvokeMessage.travelRequest,EmployeeTravelStatusService)
                  and flightDetails.flightData = clientInvokeMessage.travelRequest.flightData
                  and travelResponse1 = getService (flightDetails, AmericanAirlinesService)
                  and travelResponse2 = getService (flightDetails, DeltaAirlinesService)
                  and ((travelResponse1.flightInfo.price<= travelResponse2.flightInfo.price and travelResponse = travelResponse1)
                  or (travelResponse1.flightInfo.price> travelResponse2.flightInfo.price and travelResponse = travelResponse2))
```

Table Ⅳ. The pre- and postcondition of the whole SOFL business process

```
Precondition: clientInvokeMessage <> nil

Postcondition: flightDetails1.travelClass = getService(clientInvokeMessage.travelRequest, EmployeeTravelStatusService)
               and flightDetails1.flightData = clientInvokeMessage.travelRequest.flightData
               and flightDetails2 = flightDetails1
               and travelResponse1 = getService (flightDetails1, AmericanAirlinesService)
               and travelResponse2 = getService (flightDetails2, DeltaAirlinesService)
               and ((travelResponse1.flightInfo.price<= travelResponse2.flightInfo.price and travelResponse = travelResponse1)
               or (travelResponse1.flightInfo.price> travelResponse2.flightInfo.price and travelResponse = travelResponse2))
```

VI. CONCLUSION AND FUTURE WORK

This paper proposes the basic rules of how to transform BPEL business process into SOFL module, and demonstrate an example. After that, this paper shows how to formally review a SOFL business process.

The translation of BPEL to SOFL follows a feature-complete SOFL semantics of BPEL. Compared with BPEL workflow, the workflow described in SOFL has more advantages. The designers can review the satisfiability of each process, the viability of each data flow, the consistency between two adjacent processes, whether the process contains invalid outputs, the feasibility and consistency of invariant, and detect deadlocks. Furthermore, the designers can review whether the business process satisfies the requirements of the user.

The whole research is still ongoing, and we are still enriching the rules of transforming. In the future, we will extend SOFL further, improve the review methods of SOFL business process, and develop a tool to assist transforming BPEL workflow into SOFL module.

ACKNOWLEDGEMENT

This paper is supported by the National High-Tech Research Development Program of China (863 program) under Grant No. 2007AA01Z139

REFERENCES

[1] A. Alves, A. Arkin, S. Askary, C. Barreto, B. Bloch, F. Curbera et al. Web Services Business Process Execution Language Version 2.0. OASIS Standard, 11 April 2007, OASIS (April 2007).

[2] S. Liu, Formal Engineering for Industrial Software Development using the SOFL Method, Springer-Verlag, 428 pages, ISBN 3-540-20602-7, March 2004.

[3] S. Liu, "A Property-Based Approach to Reviewing Formal Specifications for Consistency", 16th International Conference on Software & Systems Engineering and Their Applications, Paris, France, December 2-4, Vol. 4, pp. 1/6 - 6/6, 2003.

[4] S. Liu, "Utilizing Specification Testing in Review Task Trees for Rigorous Review of Formal Specifications", Proceedings of Asia-Pacific Software Engineering Conference (APSEC03), IEEE Computer Society Press, Chiangmai, Thailand, December 10-12, pp. 510-519, 2003.

[5] S. Liu, "A Simulation Approach to Verification and Validation of Formal Specifications", Proceedings of First International Conference on Cyber World: Theory and Practice, IEEE Computer Society Press, pp. 113-120, pp. 113-120, November 6-8, 2002.

[6] S. Liu and H. Wang, "An Automated Approach to Specification Animation for Validation" Journal of Systems and Software,Elsevier Science Inc.,No. 80, pp. 1271-1285, 2007.

[7] S. Morimoto, A Survey of Formal Verification for Business Process Modeling, Lecture Notes in Computer Science 5102, pp. 514-522, 2008.

[8] Hopcroft J.E., Motwani R., and Ullman J.D., Introduction to Automata Theory, Languages, and Computation. 3rd edition. Addison-Wesley: Reading, 2006.

[9] W. Fokkink, Introduction to Process Algebra, Springer-Verlag, 1999.

[10] C.A. Petri, Kommunikation mit Automaten. PhD thesis, Rheinisch-Westfälisches Institut für Instrumentelle Mathematik an der Universität Bonn, 1962.

[11] Díaz G., Pardo J.J., Cambronero M.E., Valero V., and Cuartero F., Automatic Translation of WS-CDL Choreographies to Timed Automata, Lecture Notes in Computer Science 3670, pp. 230-242, 2005.

[12] Fu X., Bultan T., and Su J., "Analysis of Interacting BPEL Web Services". Proceedings of the 13th International Conference on the World Wide Web (WWW 2004) pp. 621-630, 2004.

[13] Salaün G., Bordeaux L., and Schaerf M., Describing and Reasoning on Web Services using Process Algebra, Proceedings of the International Conference on Web Services (ICWS 2004), pp. 43-50, 2004.

[14] Ferrara A., "Web Services: A Process Algebra Approach". Proceedings of the 2nd International Conference on Service-Oriented Computing (ICSOC 2004), pp. 242-251, 2004.

[15] H.M.W. Verbeek, W.M.P. van der Aalst, Analyzing BPEL processes using Petri nets, in: D.C. Marinescu (Ed.), Proceedings of 2nd International Workshop on Applications of Petri Nets to Coordination, Workflow and Business Process Management, pp. 59-78, June 2005.

[16] Hinz S., Schmidt K., and Stahl C., "Transforming BPEL to Petri Nets". Lecture Notes in Computer Science 3649, pp. 220-235, 2005.

[17] Y. Yang, Q. Tan, J. Yu and F. Liu, "Transformation BPEL to CP-Nets for Verifying Web Services Composition". Proceeding on the International Conference on Next Generation Web Services practices(NWeSP'05), 2005.

[18] Matjaz B. Juric, "A Hands-on Introduction to BPEL", http://www.oracle.com/technology/pub/articles/matjaz_bpel1.html, Mar. 2, 2010.