Model Checking Behavioral Specification of BPEL Web Services

Rongsheng Dong, Zhao Wei, and Xiangyu Luo

Abstract—**BPEL is a business flow language which describes the composition of web services. Since business flow may be very complex, the method of formalized analysis can help ensure the accuracy of work flow. For web services described by BPEL, in this paper we establish a formalized analysis model, provide specifications and semantics of business flow, and finally translate it into the asynchronous communication model of the model checker SPIN. Based on the formalized analysis model proposed in this paper, we analyze and verify the SAS [1], Loan and Auction protocols [2], and detect a flaw in the data flow of SAS protocol.**

Index Terms—**BPEL, Model Checking, Protocols, SPIN**

I. Introduction

Nowadays, web services have been applied in various fields, such as Business to Business, Business to Consumer and integration of enterprise applications. The Business Process Execution Language (BPEL) [2] is a standard used to specify the workflow of web service composition. Web service is a self-contained software system with its own threads of control. Basically, a web service consists of "activities" and "messages". Activities are similar to traditional programs. Messages are necessary to allow individual web services to interact with each other while maintaining their autonomy. Since this interaction may be a complex workflow, writing correct process descriptions in BPEL is not an easy task. It is necessary to verify web services with rigorous formal methods.

Model checking is a promising formal verification technique used for the verification and validation of software systems [3] [4]. The required specifications for the utilization of model checking techniques in the design of software can increase the reliability at the early stage of software development. Since the model checking techniques have been successfully applied in many cases, it has become one of the basic tools used for the development of a wide range of software. Currently more and more safety-critical software systems are developed based on web services, therefore, it is natural and also important to adopt model checking techniques for the verification of the web services composition.

In fact, some research efforts on model checking web services have already been proposed. Anupriya Ankolekar [5] uses automatic tools to verify the interaction protocols of web services described in OWL-S, provides a modeling and verification procedure for OWL-S Process, and finally verifies the death lock of the protocol and behavior property. X. Fu [1] proposes a top-down approach which specifies the set of desired conversations of a web service as a guarded automaton, which is called a "conversation protocol" and provides a tool called WSAT to translate composition of web services languages to Promela languages, and verifies the correct behavior property using SPIN at end. Shin [6] proposes the EFA model which uses automation-based model checker SPIN to verify the service flows described by BPEL [2]. Marina Mongiello [7] builds a framework that performs automatic verification of formal models of business processes through NuSMV model checker. Liao Jun [8] describes and models web services and their composition with Pi-calculus, and then verifies the validity of composition model.

Though these research efforts increase the quality of the systems based on web services to some extent, they can not meet high quality requirements of systems. In order to raise reliability of web services, we must establish an effective formalization models to accurately capture the semantics of BPEL control flow and data flow. For this purpose, in this paper we propose a method to extract behavioral specifications from the work flow described by BPEL to represent it in an agent interaction protocols [9][10] and provide a translating method to transform the formalization model into the asynchronous communication model of the model checker SPIN. To illustrate the effectiveness and efficiency of our method, we finally verify SAS [1], Loan [2], and Auction [2] protocols, and a flaw in the data flow of SAS protocol is detected.

In this paper, we utilize the agent interaction protocols to define the work flow described by BPEL. We first provide specifications and semantics of Agent Interaction Protocols, such that it could capture the attributes of control flow and data flow of BPEL. We denote each entity of web service with a single agent to help analyze the interaction of web services. The purpose of adopting this method is that it can be easily extended for the construction of multi-agent system model, which is suitable for the analysis of dynamic web services.

This paper is organized as follows: Section 2 analyzes the key techniques of BPEL web services. Section 3 provides the formalization model of web services, and defines

Manuscript received April 3, 2008. This work is supported by the National Natural Science Foundation of China under Grant No.60763004 and the Natural Science Foundation of Guangxi Province of China under Grant No. D200716 and No. 0542052.

Rongsheng Dong is with the Department of Computer Science, Guilin University of Electronic Technology, Guilin 541004, China (e-mail: dong@ guet.edu.cn).

Zhao Wei is with the Department of Computer Science, Guilin University of Electronic Technology, Guilin 541004, China (e-mail: weizhao@ mails.guet.edu.cn).

Xiangyu Luo is with the Department of Computer Science, Guilin University of Electronic Technology, Guilin 541004, China (phone: +86-773-560-1467; e-mail: ccxyluo@ guet.edu.cn).

specifications and semantics of business process. Section 4 translates the formalization model into the asynchronous communication model of SPIN. Section 5 analyzes and verifies SAS, Loan and Auction protocols.

II. Core Technique About BPEL Web Services

The purpose of this paper is to analyze the web services described by BPEL with model checking technique. BPEL web services involve the following core techniques:

(1) SOAP: SOAP defines an XML messaging protocol for basic service interoperability.

(2)WSDL: Web Services Description Language basically is an interface description language used for web service providers, which contains enough information for the clients to access. The client invokes a web service provider to use WSDL. The invocation is one-shot. WSDL does not describe global states of the provider. WSDL introduces a common grammar to describe services.

(3)BPEL: BPEL is a business process language from IBM. It extends WSDL and expresses the action of web services composition. It is composed of control flow and data flow. This also forms its two characteristics: behavior specification and message type. BPEL describes composition of web services with WSDL. A BPEL links with many partners described by WSDL. SOAP defines the message type. In BPEL, there are many control types, such as sequence, while, switch, etc. and many atomic work such as invoke, receive, reply (sending and receiving messages), and assign (updating the value of variable).

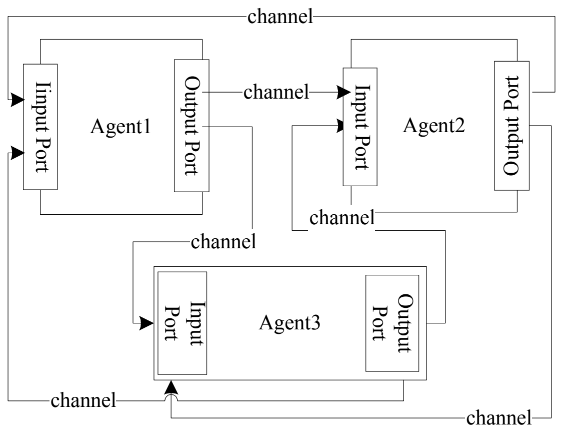

Figure 1. An abstract chart of web service composition

Based on the roles defined in the interaction between the process and its partners, BPEL defines actions and services described by WSDL (consisting of input port and output port). Fig. 1 shows an abstract chart of the composition of web services, in which Agent1, Agent2 and Agent3 denote three web services. WSDL provides the function for each web service. Web services communicate information with each other through channel. The interacting process is described in BPEL. To analyze and verify the work flow of BPEL web services, the next section will provide the formalized analysis model for such work flow.

III. Formalized Analysis Model Of BPEL Web Services

In this section we define the formalized analysis model of the work flow described by BPEL with the Interaction Protocols [9] (In defining specifications and semantics of Agent Interacting Protocols, we used Literature [11] as a reference). The protocol denotes the work flow described in BPEL. An agent denotes a partner who has two attributes: partnerLink and partnerRole. Therefore, BPEL web services can be expressed with an Interaction Protocol. The Definitions of the formalization are as follows:

Definition 1 (*Protocol*). A protocol P denotes a flow of BPEL between <process>…</process>. To define formally: A protocol $P=\langle \Theta, A, M, C, \Omega \rangle$, where Θ is the first state of BPEL flow; A is the set of agents, which is extracted from variable " <partners> <partner name="Investor"…> … </partners>"; M is the set of messages, which is extracted from WSDL files and defined with variable "<variables>" in BPEL; C is the communication chain which is extracted from "portType"; and Ω is the set of message exchange sequences. In Ω , each element is denoted by $r_i \xrightarrow[a_s, m_{k-1}]{m_k} r_j$ (when the agent r_i sends the message m_k to r_j, m_k is generated after the execution of action a_s and the prior exchange of m_{k-1}.).

Take SAS protocol for example (the process of detailed analysis is shown in Section 5):

$A=\{Investor, StockBroker, ResearchDept\}$;

$M=\{m_0, m_1, m_2, m_3, m_4, m_5, m_6, m_7, m_8\}$

Herein, A is extracted from "<partners> <partner name="Investor"…/><partner name="StockBroker" …/> <partner name=" ResearchDept " …/></partners>";

m_0 denotes message "regist"; "m_1, m_2, m_3, m_4, m_5, m_6, m_7, m_8" denote messages "ack", "cancel", "accept", "reject", "bill", "request", "terminate" and "report" respectively. They are extracted from "<variables><variable name="regist" messageType=" loandef:registMessage"/>…</ variables>".

$$\Omega_3 \in \Omega,\ \Omega_3 = \left\langle \begin{matrix} I \xrightarrow{m_0} S, S \xrightarrow[m_0, a_0]{m_3} I, S \xrightarrow[m_3 a_{12}]{m_6} R, \\ R \xrightarrow[m_6, a_{16}]{m_8} I, I \xrightarrow[m_8 a_4]{m_1} S, S \xrightarrow[m_{18}]{m_7} R \end{matrix} \right\rangle$$

Ω_3 is the intent sequence of an executed StockID. It is explained as follows: the "Investor" sends the "StockBroker" a message "regist" with only one stockID. "StockBroker" accepts the registration. The "StockBroker" will try to relay the requests to the research department, but only one stockID is sent upon each request. "ResearchDept" gives "Investor" message "report", then "Investor" sends "ack" to "StockBroker" to confirm acceptance. "StockBroker" sends "terminate" to terminate the whole service in the end.

Definition 2 (*Agent*). In our model, an agent consists of first state Θ_{r0}, the object of agent B, the sets of phases P, and the set of phase transition relations $\to$. To define formally: $\forall a \in A, a=\langle \Theta_{r0}, O, B, P, \to \rangle$, where object O is a partner which is extracted from variable " <partners>". Action B is extracted from activities "<assign>, <receive>,

<invoke>…etc". Due to the limit of space, we provide part of actions in Table 1.

In order to help readers understand the actions of agent corresponding to the activities of BPEL, we introduce some expressions in a meta-language, which is called primitives, to explain the semantics of agent. These primitives consist of functions and predicates.

Table 1. Actions

B	*meanings*	*activities*
VariableAppend	Set the initial value to variable	assign
VariableChange	The change of variable	
MessageSend	Sending messages	invoke, reply
MessageReceive	Receiving messages	receive
Endprotocol	Protocol terminating	terminate

(1) VariableAppend: This action type gives the first value of a variable. Suppose *v* is such a variable, *append*(*v*) denotes this action, value is a primitive and also a function. It returns to the value of data at a given time. Let *T* be the space of time, *d* and *t* the data and time respectively ($t \in T$), *value*(*d*, *t*) denotes this function. $value(d,t) = \varnothing$ means that the data *d* does not exist at time *t*. The VariableAppend action can be defined as follows:

$\exists(t_1,t_2) \in T \times T,\ (t_1 \neq t_2) \wedge (value(v,t_1) = \varnothing) \wedge (value(v,t_2) \neq \varnothing)$

(2) VariableChange: This action type indicates a change in the value of a variable. It is similar to action (1). Suppose *v* is such a variable, *change*(*v*) denotes this action. The VariableChange action can be defined as follows:

$\exists(t_1,t_2) \in T \times T,\ (t_1 \neq t_2) \wedge (value(v,t_1) \neq \varnothing)$
$\wedge(value(v,t_2) \neq value(v,t_1))$

(3) MessageSend: This action type means sending a message. Suppose a_i is such an action, *args*() returns the arguments of an action. *trans*() denotes sending messages. The MessageSend action can be defined as follows:

$\vee_j e_j, \forall m_j \in args(a_i), \exists k, e_k = (change(m_j) \wedge (trans(m_j) = true))$

(4) MessageReceive: This action type means receiving messages. Suppose a_i is such an action, *reception*() denotes receiving messages. The MessageReceive action can be interpreted as follows:

$\vee_j e_j, \forall m_j \in args(a_i), \exists k, e_k = reception(m_j)$

(5) Endprotocol: This action type indicates the ending of a protocol. Suppose P_{i} is a series of phases, A_{pki} the collection of a series of executable actions for phase p_{ki}, the Endphase action can be defined as follows:

$\forall p_{ki} \in P_i, (Follow(p_{ki}) = \varnothing) \vee (\ \forall a_{ki} \in A_{pki}, Unreachable(a_{ki}))$

Phase *P* is a collection of actions that have causal relationship. It is extracted from variable "<links>" in BPEL. To define formally: $\forall p \in P,\ \mathrm{p} = \langle B, \prec \rangle$, where *B* is a set of actions and $\prec$ a causal relationship. Let |*B*| be the cardinality of *B*, the Phase action can be defined as follows:

$\forall b_i, b_j \in B, b_i \neq b_j, b_i \prec b_j \leftrightarrow |B| > 1\ and\ \exists e \in Post(b_i), e \in \mathrm{Pr}\,e(b_j),$
$where\ \mathrm{Pr}\,e(x)\ returns\ the\ front\ term\ of\ x,\ and\ Post(x)\ returns\ the$
$back\ term\ of\ \ x.\ \forall b_i, b_j \in P\ and\ \ \mathrm{sup}\,pose\ b_i\ is\ in\ the\ front\ of\ b_j,$
$we\ have\ \ (b_i \prec b_j) \vee (\exists b_p, \cdots, b_k, b_i \prec b_p ..... \prec b_k \prec b_j).$

Transition relations "$\rightarrow$" means that the happening of action causes changes of phase. For example: <assign><copy><from="true"><to var="regist" part=" orderID"/></assign> means: if resist, regist.orderID=true, the state will transmit from t0 to t1; <receive operation="approve"variable="accept"/> means : if accept="approve_in", the state will transmit from t1 to t2.

IV. Translating The Formalized Analysis Model Into Promela

A. Protocol Specifications based on Promela

Now, we translate Agent Interacting Protocols *P* to a Promela program. Here, Θ is the first state; Ω is a sequence of Promela programs; *M*, *A*, and *C* are defined as follows:

(1) *M*: The type of corresponding messages "*M*" is defined as mtype in Promela (the definition of control flow in BPEL). mtype={xmsg1,xmsg2,…}. The content of corresponding messages (the definition of data flow in BPEL) can be defined with the following structure:

typedef msg_i
{*bool data*1;
int *dada*2[3];
int *data*3;......}

(2) *A*: Each agent "*A*" is defined as a global constant, such as: #define $agent_i$ 100. In this model, each agent is regarded as a Kripke structure that creates a system of finite state machine. It is expressed in a proctype type in Promela. An instance of a proctype is a process where Promela processes run concurrently and are executed non-deterministically. The regulations for the transition of state behavior of each agent are provided as follows:

$state = \Theta_0, i \geq 1, \forall \Theta_i, \Theta_{i+1}, \Theta_{i+2} \in P,$
$(state{=}{=}\Theta_i)\&\&(condition{=}{=}true) \Rightarrow atomic$
$\{chanmsgagent!agenti, xmsg_i, msg_i; j{=}data_{i+1}; state{=}\Theta_{i+1}\}$
$(state{=}{=}\Theta_i)\&\&(condition{=}{=}true) \Rightarrow$
$atomic\{chanagent_i?\mathrm{var}, parm......;$
$if::\mathrm{var}{=}{=}xmsg_i \rightarrow j{=}data_{i+1}; state{=}\Theta_{i+1};$
$::\mathrm{var}{=}{=}xmsg_{i+1} \rightarrow j{=}data_{i+2}; state{=}\Theta_{i+2}; fi;\}$

($\forall \Theta_i \in P$, Θ_i, $data_i$ are defined as constants. Suppose j,var,parm is a variable, "state" is a variable which denotes different states, and "condition" is a variable which determines if the condition is met or not.)

(3) *C*: "*C*" is transformed into Promela message channel as follow: all agents exchange information through the communication channels. Communication channels can be defined as two models. If the delay of communication is not considered and the message delivery is instantaneous, the channel can be defined as chan channel=[0] of {mtype, bool, int, …} (mtype is the messages type and "bool, int…" are the

data types of the message content). Two interactive agents share one channel, one sending messages and the other receiving messages through the corresponding communication channel. If the message delivery is not instantaneous, the channel can be defined as chan channel=[N] of {mtype, bool, int, …} ($N \geq 1$) and an additional agent—message broker agent "msgagent". In order to manage all messages as a whole, every message from $agent_i$ will be firstly sent to msgagent, and then msgagent transmits every message to the corresponding agent according to the receiving object of the message in this model. The format of message sending is: $channel!agent_i, xmsg_i, msg_i$. Notice that in this paper we adopt the second model of channel definition because many of the web services compositions involve the delay of communication and asynchronous message sending etc.

B. *Required Properties of Protocol*

This paper uses the temporal logic LTL to express the properties of protocols. LTL formulas include atomic propositions, logical operators $\wedge, \vee, \neg$ and temporal operators **X**, **G**, **U**, and **F**. Given the protocol specifications, P and each LTL property φ, and $L(P)$ are defined as the set of protocol specifications, $P \models \varphi$ if $\forall \alpha \in L(P), \alpha \models \varphi$. This is used to verify if P satisfies the property φ.

Suppose $\forall \alpha \in L(P), \alpha \cap \neg\varphi = I$, if $I = \varnothing$, the protocol satisfies the required properties; if $I \neq \varnothing$, the protocol does not satisfy the required attributes, then the corresponding counterexamples will be given by the model checker.

V. Case Studies

A. *SAS Protocol Analysis*

In this section we verify and validate a stock analysis protocol SAS [1]. We cite the analysis process of Literature [1] for SAS protocol as follows. In this protocol, three agents are involved: an *investor*, a *stock broker*, and a *research department*. First, the investor sends the stockbroker a "register" message, which consists of an "orderID", a series of stockIDs with investor's interests, and "stockprice". If the registration is accepted, the stock broker will try to relay the requests to the research department, but only one "stockID" is sent per request. In this way, a cycle forms by sending request from stockbroker to research dept, which in turn sends report to investor, and then the investor send "ack" to stockbroker. It should be noted that in this cycle, the investor has the right to cancel the whole service. If every "stockID" is processed, the stockbroker will bill the investor and then send "request" to terminate the whole service. Fig. 2 illustrates the flow chart of the protocol. In addition, we verify Loan and Auction protocols [2] specified by BPEL 2.0. Due to the limit of space, this paper only presents the analyzing process of the SAS protocol. We present part of specifications of SAS protocol as follows:

```
<process name="StockAnalysisService " […]>
   <variables> <variable name="regist"
     messageType="loandef:registMessage"/>
       […]
   </variables>
   <partners> <partner name="Investor"
     serviceLinkType="lns:InvestorLinkType"
     myRole="Investorout" partnerRole="Investorin"/>
    <partner name="StockBroker" …/>
     <partner name="ResearchDept" …/></partners>
   <flow>
     <links>
       <link name="xregist-to-xregister"/>
       <link name="xaccept-to-xrequest"/> […] </links>
       <receive  name="receive1"  partner="Investor"
          portType="ains:investoroutPT"
          operation="registerinfo" variable="regist"
          createInstance="yes">
        <source linkName="xregist-to-xregister"/>
       </receive>
       <invoke name="invokeregister" … >
       <target linkName="xregist-to-xregister"/>
       <source linkName="xregister-to-xaccept"/>
       <source linkName="xregister-to-xreject"/>
      </invoke>
       […]
   </flow>
</process>
```

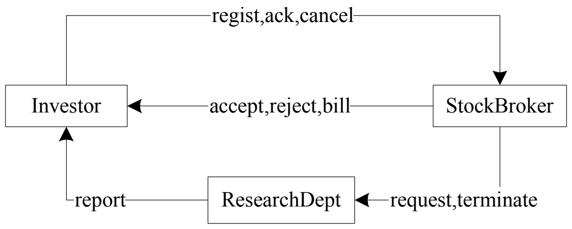

Figure 2. Flow chart of SAS protocol

B. *Description of SAS Protocol*

Four major agents are involved in this protocol: Investor, StockBroker, ResearchDept, msgmanage. These agents are described in Promela as follows:

Agent I: *proctype Investor*()

Agent S: *proctype stockbroker*()

Agent R: *proctype researchdept*()

Agent M: *proctype msg*()

All the messages sent by agent *I*, *S*, *R* will be transmitted to the communication channel of agent *M*. Agent *M* obtains information from its own channel, and then transmits it to the communication channel of the target agent. Finally, each agent obtains the required information from its own channel. The initial process in Promela is defined as follows:

init

{*run Investor*(); *run stockbroker*();

run researchdept(); *run msg*();}

First, agent *I* sends agent *S* a register message, including *n* ($1 \leq n \leq 3$) stockIDs. If the protocol is executed correctly, every stockId will send requests to agent *R* through agent *S*. After agent *I* verifies every stockid, *S* will send out the bills. Agent *I*, *S*, *R*, and *M* will run concurrently. The first three agents are the web service entities while agent *M* is an abstract entity which is responsible for sending messages. In this way, we can better simulate the asynchronous communication and the delay of data package.

Fig. 3, 4 and 5 show the finite state transition charts respectively for “Inverstor”, “StockBroker”, and “ResearchDept” in SAS protocol. We define the collection of sets of messages type as {regist, ack, cancel, accept, reject, bill, request, terminate, report}.

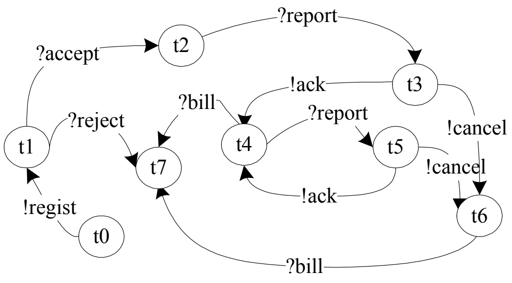

Figure 3. FSM of Investor

The set of states for “Investor” is {t0,t1,…,t7}.

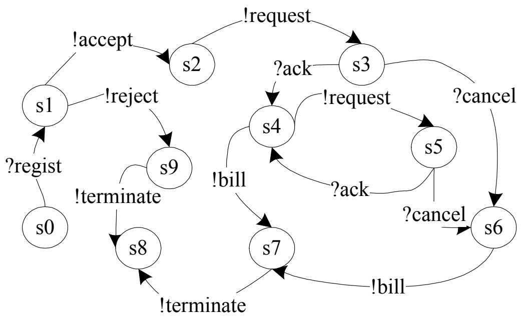

Figure 4. FSM of StockBroker

The set of states for “StockBroker” is {s0,s1,…,s9}.

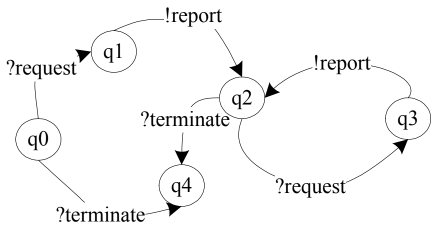

Figure 5. FSM of ResearchDept

The set of states for “ResearchDept” is{q0,q1……q4}.

C. Verification Properties of SAS protocol

Given the Promela specifications of an interactive protocol, the model checker SPIN will construct a verifier to test if the given LTL property is correct. SPIN searches the entire state of a given verification model. If the LTL property is violated, a counterexample will be provided and a path will be created. If it satisfies the Properties, it will show no errors.

(1)Safety Property: For example, the form”<>!death” is introduced to check the death lock. The experiment results show that the protocol does not have a death lock. The reachable check of the state: []<>(terminate.orderID==true) shows that the system will send message to terminate the running. This property is satisfied.

(2)Behavior Property: In SAS protocol, if the Investor does not cancel the operation, neither does the stockbroker refuse the order given by the investor. Each stockID delivered by the Investor should appear in the request information sent by the stockbroker to the researchdept. This can be expressed with LTL property as follows:

$$\text{P1: } \begin{array}{l}[]((regin.stockid[0]==2\,\&\&\,flag==1)->(<>(request.stockid==2)\\ \|<>(cancel.orderid==1)\,\|<>(reject.\,orderid\;==1)))\end{array}$$

$$\text{P2: } \begin{array}{l}[]((regin.stockid[1]==2\,\&\&\,flag==1)->(<>(request.stockid==2)\\ \|<>(cancel.orderid==1)\,\|<>(reject.\,orderid\;==1)))\end{array}$$

$$\text{P3: } \begin{array}{l}[]((regin.stockid[2]==2\,\&\&\,flag==1)->(<>(request.stockid==2)\\ \|<>(cancel.orderid==1)\,\|<>(reject.\,orderid\;==1)))\end{array}$$

Experimental result: P1 is verified to be correct. P2 and P3 are verified wrong and counterexamples are given as follows:

$regin.stockid[0]=0, regin.stockid[1]=2, regin.stockid[2]=0;$

$regin.stockid[0]=0, regin.stockid[1]=2, regin.stockid[2]=2;$

Fig. 6 illustrates the wrong track in the verification of P2:

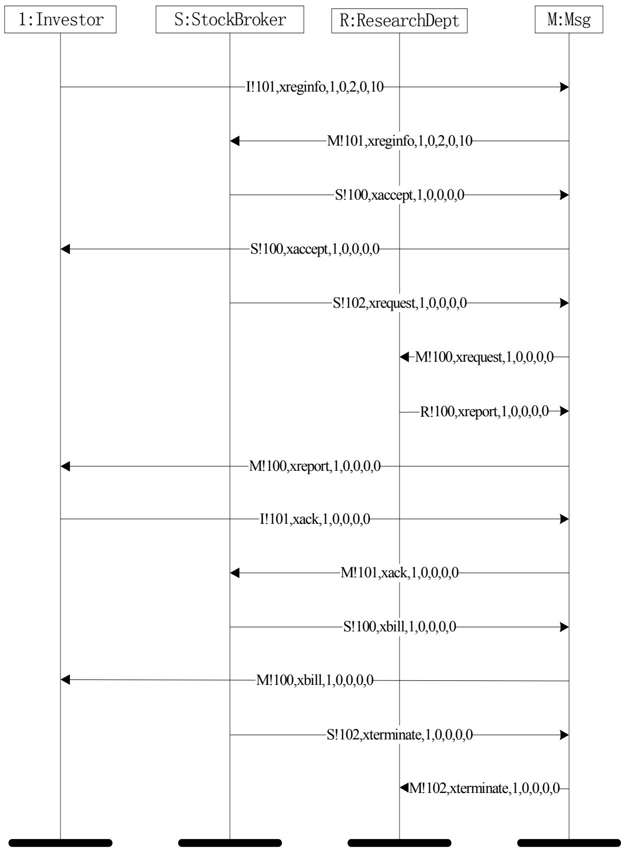

Figure 6. Wrong track chart in the verification of P2

The counterexample illustrates that the protocol executes only the first stockID while the second or the third one is not executed. The reasons for the incorrect execution of the protocol are as follows: in the business process of web services, the following question can be asked for judgment. Is the “stockID” in the register information the last “stockID”? In other words, if the “stockID”, which is sent from the present stockbroker to the researchdept after the web services are analyzed, is not the last “stockID” existing in the register information for Investor, the next “stockID” will be sent. Otherwise, it will not be sent. If the first “stockID” is the same as the last one, according to the process mentioned above, the second and the third “stockID” in the register information will not be sent to the ResearchDept. As a result, the errors occurred.

D. *Experimental Results*

To further illustrate the effectiveness and efficiency of our method, we also utilize our model to verify Loan and Auction protocol in BPEL2.0. For Loan Protocol, we verify property P1: if the Loan Request Service accepts the user, the user can request a loan less than one thousand; or the risk analysis service concludes with a low risk of loaning. P2: if the user's loan request is equal to or greater than one thousand, the user's loan request will be transferred to Risk Analysis Service to evaluate the risk. For Auction Protocol, we verify property P1: the bidder who bids the highest price will always win.

In addition, to evaluate the efficiency of our model, we adopt the method proposed in literature [1] to verify the SAS protocol. All of the experimental results are obtained in the same experimental environment: a PC running SPIN 4.3.0, with P4 3.2GHz CPU and 512MB memory. These results are listed in Table 2.

Table 2. Results of protocol verification

Source& method	Protocol& Property	State Vector (byte)	Depth	Real Time (s)	Promela Size
------ Our method	SAS: P2	556	221	0.125	293
------ Literature[1]	SAS: P2	116	851	2.287	732
BPEL2.0 Our method	Loan:P1	300	112	0.078	193
BPEL2.0 Our method	Auction: P1	432	548	35.734	228

From Table 2, we can draw the following conclusions:

(1) The time of verification is shortened. It takes only 0.125 seconds to verify SAS Protocols with our formalized analysis model. It takes 2.287 seconds in Literature [1].

(2) It requires only 293 rows to describe SAS protocol with our model but 732 rows with the translated Promela code in Literature [1].

(3) For SAS protocol, three agents are modeled respectively with three processes by using our modeling method. This modeling method can be easily extended to analyze dynamic web service, which is our next job. In literature [1], however, these three web services are modeled with a single process. We believe that if many instances of process need to be created in a web services application, the model which is built by using the modeling method in Literature [1] will increase largely.

VI. Conclusions

This paper proposes a formalized analysis method for the verification of BPEL web services composition, provides a formalized analysis model, and defines the specifications and semantics of the work flow described with BPEL. This model can capture the attributes of control flow and data flow of BPEL and is easily extended to Multi-Agent System (MAS) model to analyze the dynamic web services. This model is finally transferred to an asynchronous communication model based on SPIN. To illustrate the effectiveness and efficiency of our method, we analyze and verify the SAS protocol as well as the Loan and Auction protocols in BPEL 2.0. As for the future work, we plan to extend the formalized model proposed in this paper to MAS model so that we can further develop our symbolic model checking methods for MAS [12][13] and the corresponding model checker MCTK [13] to verify the temporal and epistemic properties in web services.

References

[1] Xiang Fu, Tevfik Bultan, and Jianwen Su. "Model Checking Interactions of Composite Web Services," UCSB Computer Science Department Technical Report (2004-05)

[2] Assaf Arkin, Sid Askary, Ben Bloch, Francisco Curbera, Yaron Goland, Neelakantan Kartha, Canyang Kevin Liu, Satish Thatte, Prasad Yendluri, Alex Yiu. Web Services Business Process Execution Language Version 2.0[Online]. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel,2005

[3] E.Clarke, O.Grumberg, and D.Peled. "Model checking,"The MIT Press, 1999.

[4] G.J.Holzmann. "the Model checker SPIN," IEEE Transaction on Software Engineering, Vol.23, 1997,pp76-95.

[5] Anupriya Ankolekar, Massimo Paolucci, and Katia Sycara. "Spinning the OWL-S Process Model-Toward the Verification of the OWL-S Process Models," In Proceedings of the ISWC 2004 Workshop on Semantic Web Services: Preparing to Meet the World of Business Applications, Hiroshima, Japan, November 2004.

[6] Shin Nakajima. "Model-Checking Behavioral Specification of BPEL Applications," Electronic Notes in Theoretical Computer Science 151 (2006) 89–105.

[7] Marina Mongiello, Daniela Castelluccia. "Modelling and verification of BPEL business processes," Proceedings of the Fourth Workshop on Model-Based Development of Computer-Based Systems and Third International Workshop on Model-Based Methodologies for Pervasive and Embedded Software (MBD/MOMPES'06).IEEE,2006.

[8] Liao Jun,Tan Hao,Liu Jin-De. "Describing and Verifying Web Service Using Pi-Calculus,"Chinese Journal of Computers,2005,(4):635-643.

[9] Wei Jun,Cheung Shing-Chi,Wang Xu. "Towards a Methodology for Forma l Design and Analysis of Agent Interaction Protocols," Wuhan University Journal of Natural Sciences, Vo l. 6 No. 122 2001,126～139

[10] Zixin Wu, Kunal Verma, Karthik Gomadam, Amit P. Sheth, John A. Miller. Process Mediation and Interaction Protocol for Web services [document/word]. http://lsdis.cs.uga.edu/~wuzixin/8351/Project%20Proposal.doc, 2006

[11] Jos´e Ghislain QUENUM2, Samir AKNINE1, Jean-Pierre BRIOT1, Shinichi HONIDEN2. A Modelling Framework for Generic Agent Interaction Protocols [Online]. http://www.informatik.uni-trier.de/~ley/db/indices/a-tree/q/Quenum:Jos=eacute=_Ghislain.html, 2006

[12] Xiangyu Luo, Kaile Su, Abdul Sattar, Mark Reynolds. "Verification of Multi-agent Systems via Bounded Model Checking," Australian Conference on Artificial Intelligence 2006: 69-78.

[13] Kaile Su, Abdul Sattar, Xiangyu Luo. "Model Checking Temporal Logics of Knowledge via OBDDs," The Computer Journal. 50(4): 403-420 (2007).

[14] Booth D, Haas H, McCabe F, Newcomer E, Champion M, Ferris C, Orchard D. Web Services Architecture [Online]. http://www.w3.org/TR/ws-arch/. 2004

[15] Alessio Lomuscio, Hongyang Qu, Marek Sergot, Monika Solanki. Verifying Temporal and Epistemic Properties of Web service Compositions [Online]. http://www.doc.ic.ac.uk/~alessio/papers.html, 2007

[16] Hai Huang, Rick A. Mason. "Model Checking Technologies for Web Services," Proceedings of the Fourth IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems and Second International Workshop on Collaborative Computing, Integration, and Assurance (SEUS-WCCIA'06).IEEE, 2006.